Índice General

Presentación

Con la aparición de este volumen 3 de los Monográficos de Rect@ se consolida el proyecto que el Consejo Editorial de la revista impulso como complemento de los números ordinarios de la revista. Esta idea inicial de editar volúmenes específicos sobre materias concretas de investigación, he tenido una buena acogida entre los miembros de la comunidad científica, tanto asociados como no de ASEPUMA, ya que las ventas de los números anteriores permiten financiar parcialmente la edición de los nuevos volúmenes.

En esta presentación quisiera agradecer el trabajo de los coordinadores (Enric Crespo, Rafa Marti y Joaquín Pacheco) de este volumen (extensiva a los de los anteriores) por la labor de captación de colaboradores, la revisión de los trabajos, la composición de los textos, etc.

Este agradecimiento también debe ser extensivo a los miembros del Consejo de Redacción (Carlos Ivorra y Vicente Liern).

Por último, recordar a todos los miembros que el futuro de estas ediciones y de la revista reside en la implicación de todos en la colaboración, aportación de ideas y sugerencias para que este tipo de publicaciones sea una referencia generalmente aceptada dentro del campo de las aplicaciones cuantitativas a la Economía y la Empresa.

Ramon Sala

Responsable de edición de Rect@.

Prefacio

Los métodos descritos en este volumen reciben el nombre de algoritmos heurísticos, metaheurísticos o sencillamente heurísticos. Este término deriva de la palabra griega *heuriskein* que significa encontrar o descubrir y se usa en el ámbito de la optimización para describir una clase de algoritmos de resolución de problemas.

En el lenguaje coloquial, optimizar significa poco más que mejorar; sin embargo, en el contexto científico la optimización es el proceso de tratar de encontrar la mejor solución posible para un determinado problema. En un problema de optimización existen diferentes soluciones, un criterio para discriminar entre ellas y el objetivo es encontrar la mejor. De forma más precisa, estos problemas se pueden expresar como encontrar el valor de unas variables de decisión para los que una determinada función objetivo alcanza su valor máximo o mínimo. El valor de las variables en ocasiones está sujeto a unas restricciones.

La existencia de una gran cantidad y variedad de problemas difíciles de optimización que aparecen en la práctica y que necesitan ser resueltos de forma eficiente, ha impulsado el desarrollo de procedimientos eficientes para encontrar buenas soluciones. Estos métodos, en los que la rapidez del proceso es tan importante como la calidad de la solución obtenida, se denominan heurísticos o aproximados.

En los últimos años se ha acuñado el término *metaheurístico*, introducido por Fred Glover en 1986 y apoyado por diferentes eventos científicos de caracter internacional, como el congreso *Metaheuristic International Conference*, o la revista *Journal of Heuristics*. El término metaheurístico establece una diferencia conceptual entre el conjunto de reglas que permiten diseñar un procedimiento de resolución heurístico y el propio procedimiento de resolución. En este sentido el prefijo *meta* indica un mayor nivel de abstracción, en cuanto que las propias reglas, denominadas procedimiento metaheurístico, no están ligadas a ningún problema específico.

Una búsqueda en internet puede darnos una medida del gran desarrollo e impacto que están teniendo estos métodos. Podemos considerar una de las metodologías más populares, los algoritmos genéticos, para realizar una prueba sencilla. Utilizando el conocido motor de búsqueda *Google* sobre el literal "genetic algorithms optimization", obtenemos más de un millón de páginas relacionadas. Esta misma búsqueda proporcionaba apenas 120.000 resultados hace cuatro años. Este simple ejercicio nos muestra el desarrollo extraordinario de los procedimien-

tos heurísticos en estos últimos años.

El desarrollo de los métodos heurísticos es tal, incluyendo la aparición de nuevas metodologías casi a diario, que excede de las posibilidades de este volumen el ofrecer una revisión exhaustiva de todos ellos. A modo de catálogo, podemos enumerar como los más establecidos los que figuran en la siguiente lista con 15 métodos (mantenemos la acepción en inglés y el acrónimo):

- Estimation Distribution Algorithms (EDA)
- Evolutionary Algorithms(EA)
- Fuzzy Adaptive Neighborhood Search (FANS)
- Genetic Algorithms (GA)
- Greedy Randomized Adaptive Search Procedure (GRASP)
- Guided Local Search (GLS)
- Heuristic Concentration (HC)
- Memetic Algorithms (MA)
- Multi-Objective Search (MOS)
- Multi-Start Methods (MSM)
- Path Relinking (PR)
- Scatter Search (SS)
- Simulated Annealing (SA)
- Tabu Search (TS)
- Variable Neighborhood Search (VNS)

En este volumen proponemos una revisión de algunos de los principales procedimientos metaheurísticos. Comenzando con una introducción a la optimización en general y los diferentes enfoques de resolución, pasamos después a revisar la búsqueda tabú, los métodos GRASP, la programación multiobjetivo, los algoritmos meméticos y la búsqueda multiarranque. Finalmente terminamos con algunas aplicaciones de estas metodologías.

Esperamos que encontréis esta recopilación interesante y os anime a entrar o a proseguir en el fascinante mundo de la optimización heurística.

Valencia, abril de 2007

Enric Crespo, Rafael Martí, Joaquín Pacheco
Coordinadores

Primera parte

Introducción

Introducción a los metaheurísticos*

M. Laguna[a] **y C. Delgado**[b]

[a] University of Colorado, Boulder,
Leeds School of Business
[b] Universidad de Burgos,
Departamento de economía Aplicada (Métodos Cuantitativos para la Economía)

1 Problemas de optimización difíciles

Optimizar es tratar de encontrar la mejor solución posible para un determinado problema. En todo proceso de optimización existen diferentes soluciones y un criterio para discriminar entre ellas. Tratamos de encontrar el valor de unas variables denominadas variables de decisión para los que la función objetivo alcanza su valor óptimo. El valor de dichas variables suele estar sujeto a unas restricciones.

Podemos encontrar una gran cantidad de problemas de optimización en la industria, en la empresa, en la economía, en la ciencia, ... Como ejemplos de problemas típicos de optimización tenemos los de localización (de servicios o actividades peligrosas), los de asignación (de personas a lugares de trabajo o similares), los de confección de calendarios (de horarios, de turnos de trabajo...), los problemas de circuitos y de distribución en planta, los de partición o cubrimiento de un conjunto (instalación de agencias de servicios que cubran una zona determinada), l os de rutas de vehículos... y otros más actuales como por ejemplo los de ingeniería y re-ingeniaría de software.

*Los autores de este trabajo agradecen la ayuda del Ministerio de Educación y Ciencia por la subvención económica para la realización de este trabajo a través del Plan Nacional de I+D, (Proyecto SEJ- 2005 08923/ECON), así como a la Junta de Castilla y León ("Consejería de Educación" – Project BU008A06).

Algunos de estos problemas son relativamente fáciles de resolver mediante diferentes métodos contrastados. Este es el caso de los problemas lineales para los que en 1947 Dantzig elaboró un método de resolución denominado método del simplex. Sin embargo la mayoría de los problemas de optimización que podemos encontrar en la práctica no se resuelven tan fácilmente. Este es el caso de los problemas de optimización combinatoria, optimización no lineal en variables continuas, optimización multiobjetivo y optimización de simulaciones, problemas que se comentan a continuación.

1.1 Optimización Combinatoria y Concepto NP

Dentro de los problemas de optimización son muy frecuentes aquellos en los que las variables sólo pueden tomar valores discretos, enteros o incluso binarios. En estos casos diremos que estamos ante problemas de optimización discreta y que, en general, se pueden formular como problemas de optimización combinatoria, expresión en la cual podemos incluir la mayoría de los problemas que tienen un número finito o numerable de soluciones alternativas.

Mas concretamente, un problema de optimización combinatoria está definido por un conjunto de soluciones factibles S (habitualmente muy numeroso), y una función $f : S \longrightarrow \mathbb{R}$. Generalmente se representan como:

$$\begin{array}{rl} \text{Minimizar (o maximizar)} & f(s) \\ \text{sujeto a:} & s \in S. \end{array}$$

La importancia de los modelos de optimización combinatoria, además del gran número de aplicaciones, estriba en que *"contiene los dos elementos que hacen atractivo un problema a los científicos: planteamiento sencillo y dificultad de resolución"*, (Garfinkel, (1985)).

Resolver un problema de Optimización Combinatoria, según Papadimitriou y Steiglitz (1982, p. 2), consiste en *"encontrar la 'mejor' solución o solución 'óptima' entre un conjunto finito o numerable de soluciones alternativas factibles"*. Se ha empleado mucho esfuerzo en investigar y desarrollar métodos para obtener soluciones óptimas o aproximadas en problemas de Optimización Combinatoria: unos basados en Programación Entera, Lineal o No Lineal, Programación Dinámica, etc. En los trabajos de Lawler (1976), Lawler, Lenstra, Rinnooy Kan and Shmoys (1985) o de Schrijver (1986) se realizan revisiones históricas del desarrollo de los diferentes métodos de solución.

A lo largo de los años se ha demostrado que muchos problemas de optimización combinatoria pertenecen a la clase de Problemas NP-completos, es decir, la solución óptima se obtiene utilizando algoritmos que emplean un tiempo de computación que crece de forma superpolinomial (normalmente exponencial) en el tamaño del problema. Por consiguiente, en muchos casos, la solución óptima no se puede obtener en un tiempo razonable (una gran variedad de este tipo de problemas se puede ver en el trabajo de Garey and Johnson (1979)).

1.2 Optimización No Lineal en Variables Continuas

Estamos ante problemas de Optimización No Lineal cuando las funciones que representan la función objetivo y las restricciones no son siempre lineales y los valores de las variables, valores no negativos, están sujetos a un conjunto de restricciones de desigualdad; esto es:

$$\begin{array}{rl} \text{Max} & f(x) \\ \text{s.a.} & g(x) \leq b \\ & x \geq 0 \end{array}$$

En el caso de que las funciones F y g_i sean continuamente diferenciables el Teorema de Kuhn-Tucker nos permite obtener las condiciones necesarias de óptimo. Básicamente, estas condiciones establecen que el gradiente de la función objetivo en x^* para que sea solución óptima, debe poderse expresar como una combinación lineal no negativa de los gradientes de las restricciones saturadas en dicho punto. Las condiciones de Kuhn-Tucker son condiciones necesarias y suficientes de óptimo local sólo para programas convexos.

Para generalizar las condiciones de Kuhn-Tucker a Programas No Diferenciables ha sido necesario desarrollar otras condiciones de optimalidad, como las *condiciones de punto de silla lagrangiano*. Tales condiciones son necesarias para casi cualquier programa matemático, y al igual que en el caso anterior son también suficientes sólo si dicho programa es convexo. Cuando el programa es diferenciable, estas condiciones más generales, como es de esperar, coinciden con las de Kuhn-Tucker.

1.3 Optimización Multiobjetivo

En la práctica real se da con frecuencia el caso de múltiples objetivos o criterios que se pretenden optimizar o satisfacer simultáneamente, por lo que usualmente entran en conflicto; dicho de otra manera, cuando nos aproximamos al óptimo de un objetivo, nos alejamos del mismo para otro objetivo. El estudio y análisis de estas situaciones ha dado origen a técnicas que se denominan Programación Multiobjetivo.

Todos los autores que tratan estas cuestiones reconocen como primer antecedente, desde un punto de vista conceptual, la aportación conocida como óptimo de Pareto, debida a dicho autor en 1896, como una parte de la Teoría del Bienestar. Sin embargo, esta idea de óptimo, tenía un carácter fundamentalmente teórico sin aplicación práctica en la toma de decisiones. Es con la obra de Charnes y Cooper (1961), cuando realmente comienza el desarrollo de las técnicas de Programación Multiobjetivo, en concreto, con el método espiral y el método de programación por objetivos. A partir de 1972, año en que se celebra la Primera Conferencia Internacional sobre la toma de Decisiones Multicriterio, Cochrane y Zeleny (Eds.)

(1973), tanto las reuniones científicas sobre la materia, como las publicaciones en revistas especializadas, no han dejado de producirse hasta nuestros días.

El planteamiento más general del problema es el siguiente:

$$\begin{array}{l} \underset{x}{max}\, F_i(x) \quad i = 1, 2, \ldots, k \\ x \in S \end{array}$$

donde x es un vector de n componentes que son las variables a las que designaremos con el término *instrumentos;* S es el *conjunto de oportunidades* y k es el número de *objetivos* que deseamos maximizar y que vienen reflejados en las funciones F.

En cuanto al proceso de decisión multicriterio, existen varios modelos, quizá el más conocido sea el de Chakong y Haimes, que distingue una fase de iniciación, la formulación del problema, modelización, análisis y evaluación, e implementación. La característica de estos modelos es que tanto en las fases de formulación, como de evaluación, intervienen frecuentemente elementos subjetivos.

Entre los múltiples criterios que pueden darse a la hora de clasificar las técnicas multiobjetivo, escogemos la que hace referencia a la relación entre el analista y el decisor; según sea esa relación se originan distintas técnicas de solución:

1. Técnicas generadoras: Se generan un conjunto de alternativas eficientes siendo el flujo de información del analista al decisor.

2. Técnicas con información a priori: La dirección de la información es del decisor al analista; la más conocida es la *Programación por Objetivos.*

3. Técnicas interactivas: El flujo de información va en ambas direcciones.

Entre los algoritmos de resolución los más sencillos son los basados en la representación de las metas múltiples mediante una sola función objetivo. En el método de ponderación, se forma una sola función objetivo como la suma de los valores asignados de las funciones que representan las metas del problema. En el método por prioridades se empieza por determinar las prioridades de las metas en orden de importancia; el modelo es entonces optimizado utilizando una meta a la vez y de tal manera que el valor óptimo de una meta de prioridad más elevada no se degrade por una meta de prioridad más baja.

1.4 Optimización de Simulaciones

La simulación es una técnica que enseña a construir el modelo de una situación real permitiéndonos a su vez la realización de experimentos con ese modelo.

En concreto *"simulación es una técnica numérica para conducir experimentos en una computadora digital, los cuales requieren ciertos tipos de modelos lógicos y matemáticos, que describen el comportamiento de un negocio o de otros tipos de sistemas en periodos extensos de tiempo"* (Naylor y otros, 1982).

La utilización de modelos de simulación ha sido una técnica muy empleada para aproximarnos a problemas muy complicados de tratar analíticamente. En los modelos de simulación, habitualmente, no se puede obtener la función objetivo en función de las variables o parámetros de entrada de forma explícita mediante una fórmula cerrada. La falta de una función objetivo de forma explícita que relacione el objetivo (costes, tiempos) a optimizar con los parámetros de entrada hace que sea necesario ejecutar la simulación para cada combinación de parámetros; a poco complejo que sea el problema el número de combinaciones posibles de los parámetros puede ser elevado y el tiempo de computación excesivo. Además si, como suele pasar en muchos casos, existen componentes aleatorios en el modelo, entonces se debe replicar varias veces la simulación para obtener una estimación aproximada de la función objetivo, lo que supone un incremento notable del tiempo de computación.

La optimización de simulaciones es una técnica que trata de buscar los valores óptimos o cercanos al óptimo de los parámetros de entrada. Esta técnica es relativamente nueva, y a pesar de los inconvenientes de su implementación, parece que tendrá un impacto considerable en la práctica de la simulación en el futuro, particularmente cuando los ordenadores lleguen a ser más rápidos.

2 Enfoques Básicos de Solución

Como ya se ha comentado muchos problemas de Optimización Combinatoria pertenecen a la clase de Problemas *NP-Hard,* es decir, para encontrar el óptimo necesitan un tiempo de computación que crece de forma (al menos) exponencial en el tamaño del problema. En estos casos debemos decidir entre encontrar la solución óptima a costa de emplear un tiempo de computación muy elevado o encontrar una buena solución en un tiempo de computación razonable.

En el primer caso se han de utilizar algoritmos Optimos o Exactos, mientras que en el segundo se utilizarán algoritmos de Aproximación o Heurísticos, como los de búsqueda local, algoritmos constructivos, búsqueda incompleta, etc.

Por otro lado existen situaciones en las que puede ser conveniente encontrar una *buena* solución en lugar de la mejor de ellas. Las más evidentes son las siguientes:

- En muchos problemas reales los datos no son exactos sino sólo una buena aproximación. Es evidente que insistir aquí en un costoso procedimiento exacto carece de sentido.
- Existen modelos, muy frecuentemente, en los cuales es imposible incluir todas las consideraciones que afectan al problema ya que algunas de ellas son difícilmente objetivizables y modelizables. En estos casos más que una solución exacta lo que necesita el decisor son varias *buenas* soluciones entre las que optar en función de sus propios criterios.
- Cuando hay que tomar decisiones en tiempo real, caso de autómatas o programaciones de trabajo sobre la marcha, lo que interesa es una *buena* decisión pero inmediata.

Todos estos casos justifican la utilización de métodos aproximados si éstos roporcionan buenas soluciones en un tiempo razonable. Por lo general los algoritmos heurísticos tienen la ventaja añadida de que pueden ser más fácilmente adaptables para solucionar modelos más complejos. Hay que señalar que los diferentes tipos de algoritmos dependen de cada problema concreto, ya que en su diseño se intentan aprovechar las características específicas de dichos problemas.

2.1 Métodos Exactos

Como ya se ha comentado permiten encontrar la solución óptima a un problema. La enumeración explícita de todo el conjunto de soluciones, aunque esté acotado, suele ser excesiva, ya que este puede ser de gran tamaño incluso en problemas con pocas variables; por tanto se aplican métodos que abrevian dicha búsqueda. La mayoría de los algoritmos están basados en procesos de *Ramificación y Acotamiento (Branch & Bound)* consistentes en lo siguiente:

- Inicialmente, partir del conjunto de todas las soluciones; dividir dicho conjunto en dos subconjuntos o ramas; calcular y asignar para cada rama una cota inferior, si estamos minimizando (superior si estamos maximizando), del valor de la función objetivo en ese conjunto de soluciones.
- Elegir una de las ramas o subconjuntos según algún criterio y realizar de nuevo el paso anterior. Repetir este proceso hasta llegar a una única solución.
- Continuar el proceso en las ramas que queden sin explorar si la cota inferior asociada es menor que el valor de la mejor solución encontrada hasta ese momento. De esta forma se evita realizar exploraciones innecesarias.

Los criterios de ramificación y elección de la siguiente rama o subconjunto que se explora, y la forma de determinar las cotas, dependen de la estructura y características propias de cada problema concreto, y son importantes para aumentar la

eficacia de estos algoritmos. El valor de la solución de la relajación continua del problema, en el conjunto de soluciones correspondiente, se utiliza habitualmente como cota inferior. Los conjuntos de soluciones se dividen según los valores de una determinada variable, y se elige como nueva rama a explorar la de menor cota inferior.

Menos eficiente es la resolución usando exclusivamente técnicas de Programación Dinámica, sin embargo éstas pueden ser útiles para hallar cotas que después sean introducidas en algoritmos de tipo Branch & Bound.

Otros métodos desarrollados son el algoritmo de *Plano de Corte* de Gomory, muy aceptado para determinados problemas o el de *Descomposición* de Benders. Veamos como funciona el primero de ellos. Al igual que en los procesos de Ramificación y Acotamiento en el algoritmo de Plano de Corte de Gomory se utiliza la solución continua óptima de PL. En este caso se modifica el espacio de la solución añadiendo sucesivamente restricciones especialmente construidas llamadas cortes. En la tabla óptima del simplex del problema relajado se añade el corte generado a partir de un renglón elegido de forma arbitraria, de entre aquellos cuya variable no es entera. A partir de dicha tabla se calcula la solución óptima factible, y se repite el proceso hasta que todas las variables sean enteras.

La idea gráfica del algoritmo de Plano de Corte de Gomory se muestra en las figuras 1 y 2. Los cortes añadidos no eliminan ninguno de los puntos factibles pero deben atravesar por lo menos un punto entero factible o no factible.

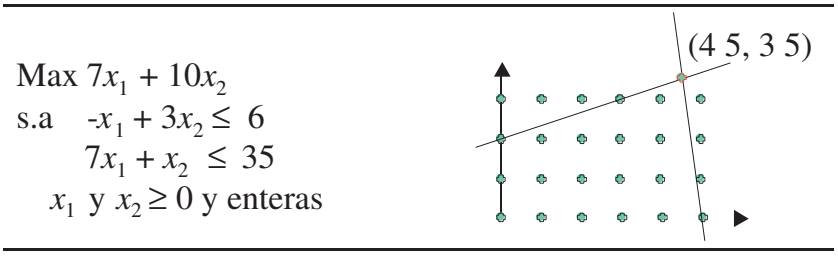


Fig. 1 Formulación del problema y solución considerando variables contínuas

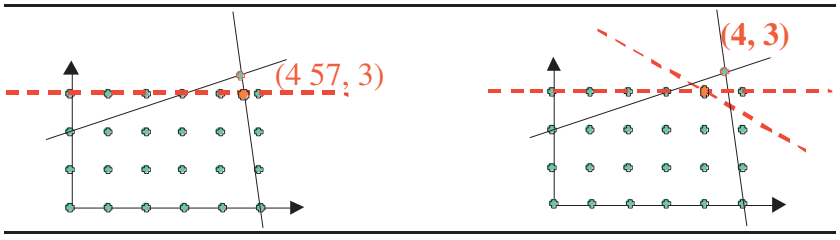


Fig. 2 Cortes realizados y solución asociada en cada caso

En cualquier caso, el tiempo de computación utilizado, suele ser alto. Por esta razón se da mucha importancia, especialmente en problemas combinatorios, al diseño y uso de algoritmos heurísticos, esto es, algoritmos que no garantizan la obtención de la solución óptima, pero sí una buena solución en un tiempo mucho menor.

2.2 Heurísticos de Construcción

Existen diferentes *tipos de heurísticos,* según el modo en que buscan y construyen sus soluciones. En los heurísticos de construcción se van añadiendo paulatinamente componentes individuales a la solución, hasta que se obtiene una solución factible.

El más popular de estos métodos lo constituyen los algoritmos golosos o devoradores (greedy), que construyen la solución seleccionando en cada paso la mejor opción. Otras versiones escogen de forma totalmente aleatoria en cada paso el componente que se añadirá a la solución. Las versiones que en general han demostrado dar mejores resultados son las que introducen sólo cierto grado de aleatoriedad: primero seleccionan un conjunto de buenas opciones; posteriormente escogen el siguiente componente de la solución de forma aleatoria en dicho conjunto.

2.3 Búsqueda Local

Dentro de los métodos aproximados son de uso muy frecuente en la optimización combinatoria los métodos de Búsqueda Local. Dichos métodos se basan en la idea de explorar las soluciones 'vecinas' de aquella que tenemos en un momento dado. Por tanto para diseñar un procedimiento de búsqueda local, es necesario previamente definir un *vencidario* o *entorno* $N(s)$, $\forall s \in S$, es decir:

$N(s) =$ Conjunto de soluciones vecinas de s (a las que se llega por un pequeño movimiento o cambio en s)

Se trata sencillamente de pasar de una solución a otra vecina que sea mejor. El proceso se acaba cuando no hay mejora posible en el conjunto de soluciones vecinas. Estas técnicas pueden ser más sencillas o más sofisticadas dependiendo de la estructura vecinal que se defina.

El siguiente procedimiento describe este proceso:

Seleccionar una solución inicial $s_0 \in S$.
Repetir
    Seleccionar $s \in N(s_0)$ tal que $f(s) < f(s_0)$ por un método preestablecido
    Reemplazar s_0 por s
hasta que $f(s) \geq f(s_0)$, $s \in N(s_0)$.

Las formas más usuales de seleccionar la solución vecina son explorar en todo el vecindario y tomar la mejor solución según el valor de la función objetivo *(mayor descenso),* o buscar y seleccionar la primera que mejora la solución actual *(primer descenso).*

El inconveniente de esta estrategia siempre *descendente* es que en la mayoría de los casos se convergen a mínimos locales que no son globales, y además suelen estar muy lejos del óptimo global. Para ilustrar este problema considérese la siguiente gráfica:

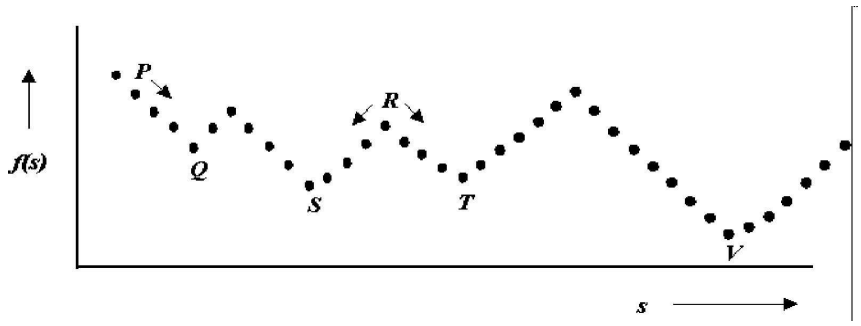


Fig. 3 Ejemplo de estrategia descendente

En esta figura se muestra un sencillo ejemplo de una función en la que para cada solución se consideran dos soluciones vecinas, representadas por el punto que tiene a la izquierda y el que tiene a la derecha. Una estrategia descendente siempre se dirige hacia el *fondo del valle* que contiene al punto de salida. Por ejemplo, si la solución inicial es P, siempre se finalizará en el punto Q.

Para evitar este problema, se han sugerido algunas posibles soluciones: repetir el algoritmo con diferentes puntos de partida, o aumentar el *vecindario.* Por desgracia, ninguna de estas variantes ha resultado ser totalmente satisfactoria. En la figura anterior se observa que para evitar que la búsqueda se quede atrapada en mínimos locales de mala calidad, se debería permitir algunos movimientos *hacia arriba*, de forma controlada, aunque empeoren momentáneamente el valor de la solución actual. De esta forma se podrían visitar óptimos locales de mejor calidad e incluso el óptimo global.

En definitiva la búsqueda local, como tal, es una búsqueda *ciega* ya que el único criterio para aceptar una solución es que reduzca el valor de la función objetivo. Por tanto no utiliza ninguna información recogida durante la ejecución del algoritmo y depende de una manera muy estrecha de la solución inicial y del mecanismo de generación de entornos. Para evitar quedar atrapado en un óptimo local y poder continuar buscando soluciones mejores en todo el espacio es por lo que se han creado diversas estrategias incluidas en los metaheurísticos.

3 Metaheurísticos

Los metaheurísticos son el desarrollo más reciente, entre los métodos aproximados, para resolver complejos problemas de optimización combinatoria que aparecen en la empresa, la economía, la industria, la ingeniería y muchas otras áreas. Se han desarrollado vertiginosamente desde principios de los 80 para resolver una gama muy variada de problemas.

Una definición de Metaheurístico dada por Osman y Kelly (1996) es la siguiente:

> *"Un metaheurístico es un procedimiento iterativo, con una estructura y una reglas generales de funcionamiento que lo caracterizan, que guía un método (normalmente un heurístico) subordinado combinando inteligentemente diversos conceptos para explorar los espacios de búsqueda utilizando estrategias aprendidas para conseguir soluciones quasi- óptimas de manera eficiente".*

Según Glover y Laguna (1997):

> *Metaheurística se refiere a una estrategia maestra que guía y modifica otras heurísticas para producir soluciones más allá de aquellas que normalmente se generan en una búsqueda de óptimos locales."*

Utilizan conceptos derivados de la inteligencia artificial, la biología, las matemáticas... para mejorar su eficacia. En general, tratan de explotar una colección de ideas *sensatas* para ir mejorando la calidad de las soluciones.

Son procedimientos iterativos que disponen de mecanismos de parada fijados por el usuario como pueden ser la cantidad de iteraciones efectuadas, el número de iteraciones sin mejorar o haberse acercado suficientemente al óptimo (si se dispone de una cota del mismo), etc. Los *criterios de parada* son absolutamente necesarios en este tipo de procedimientos ya que continúan la exploración después de haber obtenido un óptimo local, y sin ellos el proceso seria interminable.

Aunque es posible encontrar convergencias teóricas al óptimo global para algunos metaheurísticos bajo determinadas hipótesis, estas hipótesis no se verifican en la mayoría de las aplicaciones prácticas. Por tanto, aunque pierden la posibilidad de garantizar soluciones óptimas, los metaheurísticos han obtenido éxitos a la hora de conseguir *buenas* soluciones en una amplia gama de aplicaciones en muy diversas áreas.

Además tienen otra gran ventaja. Dada la sencillez de sus elementos básicos y la importancia de sus aspectos intuitivos pueden ser implementados y utilizados por personas sin una formación específica en matemáticas de alto nivel. Aunque hay que tener en cuenta que a mayor conocimiento de técnicas de investigación operativa, mayor capacidad de recursos para abordar los problemas.

A la hora de describir un metaheurístico las características principales que se han de comentar son las siguientes:

- Algoritmos determinísticos
- Algoritmos aleatorios
- Uso de memoria
- Algoritmos poblacionales
- Uso de movimientos por entornos
- Basados en procesos físicos, biológicos, inteligencia artificial, ..

Tradicionalmente, dos de estas características se solían utilizar para dividir los metaheurísticos, en general, en dos grandes grupos: metaheurísticos basados en Población o Algoritmos Evolutivos, y metaheurísticos basados en Búsqueda por Entornos. Sin embargo en la actualidad la mayoría de los métodos que se implementan hoy en día son híbridos, de forma que los métodos basados en poblaciones incluyen búsquedas locales que por definición se realizan por medio de entornos. En los dos puntos siguientes (3.2 y 3.3) se comenta su funcionamiento y se explican algunos de estos metaheurísticos.

También se puede hablar de métodos que utilizan decisiones sistemáticas (muchas veces basados en memoria) y métodos basados en decisiones aleatorias. Al igual que en el caso anterior es difícil encontrar implementaciones "puras" ya que la mayoría de los procedimientos incluyen ambos tipos de estrategias (sistemáticas y aleatorias o pseudos aleatorias). En el punto 3.4 se estudia el funcionamiento básico de tres metaheurísticos basados en muestreo aleatorio. En este caso, nos centramos en metaheurísticos que no utilizan memoria. Sus decisiones son todas al azar sin tener en cuenta las características del problema que trata o los resultados obtenidos con anterioridad.

Recientemente se han desarrollado estrategias heurísticas que eligen entre heurísticos para resolver problemas de optimización. Estos métodos se denominan *hiperheurísticos* y su objetivo principal es el de diseñar estrategias de programación generales que puedan ser aplicadas a diferentes problemas. Habitualmente los algoritmos metaheurísticos suelen funcionar bien para los problemas para los que se han diseñado, pero no para todo tipo de problemas. Cuando se pretenden aplicar a un tipo de problema diferente hay que realizar modificaciones en el método, a menudo numerosas y costosas. En el caso de los hiperheurísticos esto no sería necesario. Otra característica propia de los hiperheurísticos es que mientras un metaheurístico modifica las soluciones directamente, un hiperheurístico lo hace indirectamente, por medio de un operador (un heurístico de bajo nivel). Por supuesto los hiperheurísticos pueden ser metaheurísticos y de hecho normalmente lo son.

3.1 Métodos Basados en Búsqueda por Entornos

La *Búsqueda por Entornos* trata de superar los inconvenientes de la Búsqueda Local: dependencia de la solución inicial y convergencia a mínimos locales que no son globales. Para salir de esos mínimos locales deberían permitirse movimientos que empeoren momentáneamente la solución actual. Como se verá más adelante, algunos métodos heurísticos, permiten estos movimientos "hacia arriba" de forma controlada, mejorándose en muchos casos el valor de la solución final. Otros repiten el algoritmo con diferentes puntos de partida o consideran una estructura vecinal más compleja (modifican el vecindario).

Tabu Search (TS)

Según Glover F. (1996) la palabra *Tabu* se refiere a: *"... un tipo de inhibición por connotaciones culturales o históricas que puede ser superado en determinadas condiciones..."*.

Tabu Search (Búsqueda Tabú) dada a conocer por Glover F. (1989) y (1990-a), es un procedimiento metaheurístico utilizado con el fin de guiar un algoritmo heurístico de búsqueda local para explorar el espacio de soluciones más allá de la simple optimalidad local y obtener soluciones cercanas al óptimo. Se han publicado numerosos artículos y libros para difundir el conocimiento teórico del procedimiento; en Glover F. y Laguna M. (1997) y (2002) pueden encontrarse amplios tutoriales sobre Búsqueda Tabú que incluyen todo tipo de aplicaciones.

Al igual que la búsqueda local, la búsqueda tabú en su diseño básico, constituye una forma agresiva de búsqueda del mejor de los movimientos posibles a cada paso; sin embargo, también permite movimientos hacia soluciones del entorno aunque no sean tan buenas como la actual, de forma que se pueda escapar de óptimos locales y continuar la búsqueda de soluciones aún mejores (ver figura 4).

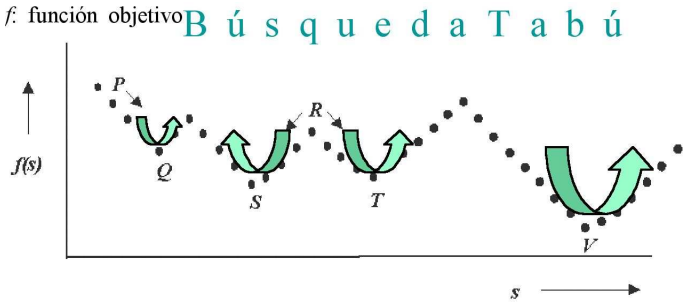


Fig. 4 Procedimiento de Búsqueda Tabú.

Simultáneamente para evitar ciclos, los últimos movimientos realizados son declarados tabú durante un determinado número de iteraciones, utilizando las denominadas *restricciones tabú.* Dicha condición tabú puede ser ignorada bajo determinadas circunstancias dando lugar a los llamados *criterios de aspiración.* De esta forma se introduce cierta flexibilidad en la búsqueda.

Variable Neighborhood Search (VNS)

Variable Neighborhood Search (Búsqueda en Entorno Variable) es una técnica metaheurística propuesta y descrita en los trabajos de Mladenovic (1995), Mladenovic y Hansen (1997) y Hansen y Mladenovic (1998). La idea básica es combinar la aplicación de un procedimiento de búsqueda local con un cambio sistemático del entorno de búsqueda. El algoritmo aplica la búsqueda local en alguna solución del entrono de la mejor solución obtenida hasta el momento (solución actual). Si no se consigue mejorar esta solución actual se considera un entorno mayor. Cuando se obtiene una solución mejor se reinicia el proceso. Intenta explotar la idea de que los mínimos locales tienden a concentrarse en unas pocas regiones. Un tutorial más recientes se encuentran en Hansen y Mladenovic (2003).

3.2 Métodos Basados en Poblaciones

Los *Algoritmos Evolutivos* o algoritmos basados en poblaciones se inspiran en los principios básicos de la evolución de los seres vivos, y modifican dichos principios para obtener sistemas eficientes para la resolución de diferentes problemas. Un Algoritmo Evolutivo es un proceso estocástico e iterativo que opera sobre un conjunto P de *individuos* que constituyen lo que se denomina *población;* cada individuo contiene uno o más cromosomas que le permiten representar una posible solución al problema, la cual se obtiene gracias a un proceso de codificación/decodificación. La población inicial es generada aleatoriamente o con la ayuda de algún heurístico de construcción. Cada individuo es evaluado a través de una función de adecuación *(fitness).* Estas evaluaciones se usan para predisponer la selección de cromosomas de forma que los superiores (aquellos con mayor evaluación) se reproduzcan más a menudo que los inferiores.

El algoritmo se estructura en tres fases principales que se repiten de forma iterativa, lo que constituye el *ciclo reproductivo básico* o *generación.* Dichas fases son: selección, reproducción y reemplazo.

Genetic Algorithms (GA)

A finales de los 60 y principios de los 70 distintos investigadores trataron de trasladar los principios de la Evolución al campo de la algoritmia, dando lugar a lo que tradicionalmente se conocía como *Evolutionary Computation* y que ahora se llama *Evolutionary Algorithms.* Como resultado de esta investigación

se originaron diferentes modelos que pueden agruparse en tres grandes familias: *Evolutionary Programming (Programación Evolutiva), Evolutions Strategies (EE, Estrategias de Evolución)* y *Genetic Algorithms (GA, Algoritmos Genéticos).*

Filosóficamente los tres métodos sólo difieren en el nivel de detalle en que simulan la evolución. A nivel algorítmico difieren en la forma en que representan las soluciones y los operadores que usan para modificarlas. La *Programación Evolutiva* tiene su origen en el trabajo de Fogel L.J. y otros (1966), y ponen un especial énfasis en la adaptación de los individuos más que en la evolución del material genético de éstos. Las *Estrategias de Evolución* comenzaron a desarrollarse en Alemania. Su objetivo inicial era servir de herramienta para la optimización de parámetros en problemas de ingeniería. Al igual que la Programación Evolutiva con la que se halla estrechamente emparentada, basa su funcionamiento en el empleo de un operador de reproducción asexual o de mutación, especialmente diseñado para trabajar con números reales. En cuanto a los *Algoritmos Genéticos* son probablemente el representante más conocido de los algoritmos evolutivos, y aquellos cuyo uso está más extendido. Fueron concebidas originalmente por John Holland y descritas en el ya clásico *Adaptation in Natural and Artificial Systems* (Holland J. (1975)). Funcionan mediante la creación en una computadora, de una población de individuos representados por los cromosomas, que son en esencia un conjunto de cadenas de caracteres análogos a los cromosomas de cuatro bases de nuestro ADN.

Históricamente, el término evolutivo se ha asociado con algoritmos que usaban solamente selección y mutación, mientras que el término genético ha sido asociado a algoritmos que usan selección, mutación, cruce y una variedad de otros mecanismos inspirados en la naturaleza (Goldberg D.E. (1994)). La principal característica de los GAs es el uso del operador de recombinación o cruce como mecanismo principal de búsqueda: construye descendientes que poseen características de los cromosomas que se cruzan. Su utilidad viene dada por la suposición de que diferentes partes de la solución óptima pueden ser descubiertas independientemente y luego ser combinadas para formar mejores soluciones. Adicionalmente se emplea un operador de mutación cuyo uso se considera importante como responsable del mantenimiento de la diversidad.

La premisa que guía los GAs es que pueden resolverse problemas complejos simulando la evolución en un algoritmo programado por ordenador. La concepción de John Holland es que esto ocurre mediante algoritmos que manipulan strings binarios llamados *cromosomas.* Como en la evolución biológica, la evolución simulada tiene el objetivo de encontrar buenos cromosomas mediante una manipulación ciega de sus contenidos. El término ciego se refiere al hecho de que el proceso no tiene información sobre el problema que intenta resolver.

Los primeros diseños de Holland fueron simples, pero probaron ser efectivos para solucionar problemas considerados difíciles en aquel tiempo. El campo de los GAs se ha desarrollado desde entonces, principalmente como resultado de las

innovaciones en 1980 al incorporar más diseños elaborados con el propósito de resolver problemas en un amplio rango de escenarios prácticos.

Los componentes que han de considerarse a la hora de implementar un GA son los siguientes:

- Una representación, en términos de "cromosomas", de las configuraciones de cada problema: método de codificación del espacio de soluciones en cromosomas.
- Una manera de crear las configuraciones de la población inicial.
- Una función de evaluación que permita ordenar los cromosomas de acuerdo con la función objetivo: medida de la bondad o función fitness.
- Operadores genéticos que permitan alterar la composición de los nuevos cromosomas generados por los padres durante la reproducción.
- Valores de los parámetros que el algoritmo genético usa (tamaño de la población, probabilidades asociadas con la aplicación de los operadores genéticos).

Scatter Search (SS)

Scatter Search (Búsqueda Dispersa) se caracteriza por el uso de un conjunto de soluciones, denominado *Conjunto de Referencia,* que es actualizado durante el proceso. El modo en el que combina soluciones y actualiza el conjunto de soluciones de referencia usadas para combinar conjuntos, aparta esta metodología de otros enfoques basados en población. Quizás el método más cercano a SS dentro del área de Evolutionary Algorithms es Evolution Strategies ya que dichos métodos utilizan un esquema determinista de selección y una representación de soluciones que es natural al problema en lugar de la representación genética para cada individuo. Estas estrategias, en contraste con los GA's, no simulan la evolución al nivel genético.

Como se comenta en los trabajos de Glover F. (1998) y Campos V. y otros (2001) el enfoque de combinación de soluciones para crear nuevas soluciones se originó en los 60. La estrategia combinatoria se diseñó con la confianza de que la información sería explotada más efectivamente de forma integrada que tratándola aisladamente (Crowston W.B. y otros (1963); Fisher H. y Thompson G.L. (1963)).

Scatter Search opera en un conjunto de referencia (*reference set,* RS) combinando soluciones para crear unas nuevas. El conjunto de referencia puede evolucionar como se ilustra en la figura 5, cuando se crean nuevas soluciones de una combinación lineal de otras dos o más soluciones. Esta figura asume que el conjunto de referencia original de soluciones consta de los círculos etiquetados como A, B y C. Después, una combinación no convexa de las soluciones de referencia A y B crea la solución 1. En concreto se crean un número de soluciones en el segmento definido por A y B; sin embargo sólo la solución 1 se introduce en el

conjunto de referencia. De modo similar, las combinaciones convexas y no convexas del conjunto de referencia original y la solución recién creada, crea los puntos 2, 3 y 4. El conjunto de referencia completo mostrado en dicha figura consta de 7 soluciones.

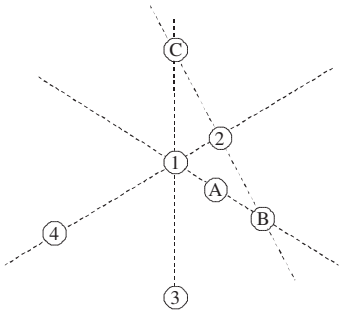


Fig. 5 Conjunto de Referencia (tomado de Laguna M. (2002)).

El conjunto de referencia de soluciones en búsqueda dispersa tiende a ser pequeño, a diferencia de la población de los algoritmos genéticos. En algoritmos genéticos se eligen aleatoriamente dos individuos de la población y se aplica un mecanismo de cruce o combinación para generar uno o más hijos. Un tamaño de población típico en algoritmos genéticos consta de 100 elementos, que son probados aleatoriamente para crear combinaciones. En contraste, SS elige dos o más elementos del conjunto de referencia de forma sistemática con el propósito de crear nuevas soluciones. Ya que el proceso de combinación considera al menos todos los pares de soluciones del conjunto de referencia, en la práctica se necesita trabajar con conjuntos de pocos elementos. Normalmente el conjunto de referencia en búsqueda dispersa tiene 20 soluciones o menos. En general, si el conjunto de referencia consta de b soluciones, el procedimiento examina aproximadamente $(3b-7)b/2$ combinaciones de cuatro tipos diferentes (Glover F. (1998)). El tipo básico consta de combinaciones de dos soluciones; el siguiente tipo combina tres soluciones y así seguimos con cuatro o más soluciones dependiendo del problema. La limitación del campo de búsqueda a un grupo selectivo de tipos de combinación puede usarse como un mecanismo de control del número de combinaciones posibles en un conjunto de referencia dado.

Las soluciones en SS no solo se pueden combinar utilizando combinaciones lineales. Así una extensión natural del método es utilizar *Path Relinking* (Re-encadenamiento de Trayectorias) para combinar soluciones. El Re-encadenamiento de Trayectorias se basa en el hecho de que entre dos soluciones se puede trazar un camino que las una, de modo que las soluciones en dicho camino contengan atributos de las iniciales. Para generar los caminos es necesario seleccionar movimientos que cumplan los siguientes objetivos: empezando por una solución inicial x', los movimientos deben introducir progresivamente los atributos de la solución

guía x'' (o reducir la distancia entre los atributos de estas soluciones). Los papeles de ambas soluciones son intercambiables; además cada solución puede moverse hacia la otra como una manera de generar combinaciones.

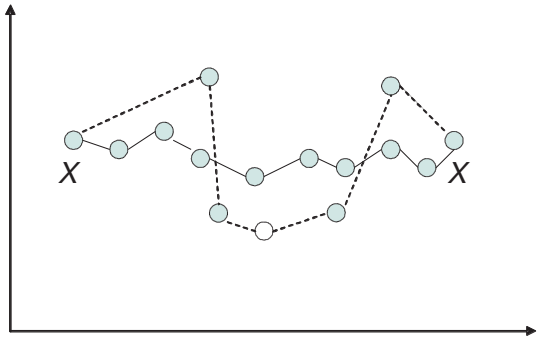


Fig. 6 Trayectoria Path Relinking (- - -) con solución mejor

Utilizando PR se genera por lo tanto un camino que une dos soluciones seleccionadas x' y x'' produciendo una secuencia de soluciones

$$x' = x(1), x(2), \ldots, x(r) = x''.$$

En este camino es posible por ejemplo encontrar soluciones mejores que las soluciones inicial y guía (figura 6) y además estos caminos son habitualmente "más directos" que los encontrados por otras estrategias para unir dichas soluciones (figura 7).

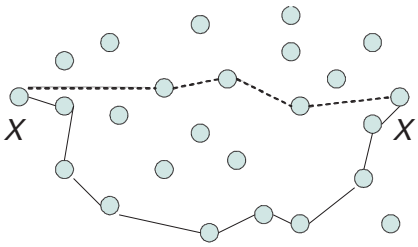


Fig. 7 Trayectoria Path Relinking (- - -) con camino más corto

Particle Swarm Optimization (PSO)

PSO originalmente fue concebido para simular de un sistema social simplificado; sin embargo se vio que el modelo podía ser usado como optimizador. PSO es una técnica de optimización estocástica poblacional desarrollada por Kennedy J. y Eberhart R. (1995), e inspirada en el comportamiento de organismos tales como las bandadas de pájaros. Comparte algunas similitudes con Algoritmos

Evolutivos tales como los Algoritmos Genéticos. Comienza con un conjunto de soluciones aleatorias y busca la solución óptima actualizando generaciones. Sin embargo a diferencia de los GA, PSO no utiliza operadores evolutivos como el cruce o la mutación.

PSO imita el comportamiento de una bandada de pájaros en busca de alimentos. Un grupo de pájaros busca comida en un determinado área. Se supone que sólo hay una pieza de comida en el área de búsqueda. Los pájaros no saben dónde se encuentra la comida, pero si saben en cada iteración a qué distancia está. La estrategia será seguir al pájaro que más cerca se encuentre de la comida. De esta forma, las posibles soluciones de PSO (los pájaros) llamadas partículas "vuelan" a través del espacio de soluciones cambiando su posición y velocidad en función de su propia experiencia y de la experiencia de las vecinas. La "velocidad" de cada partícula se ve modificada por una fórmula muy sencilla que tiene dos elementos: uno que impulsa a la partícula hacia la mejor posición (solución al problema) en la que esa partícula ha estado durante la búsqueda y otro que impulsa a la partícula hacia la mejor posición encontrada por todas las partículas en la búsqueda. La implementación es muy sencilla ya que cada partícula sólo tiene que recordar cuál es la mejor posición en la que ha estado y cuál es la mejor posición encontrada por todas las partículas.

PSO y Path Relinking son muy parecidos en el aspecto de que las soluciones guías en PR juegan el mismo papel que las posiciones hacia las cuales las partículas son impulsadas en cada paso del PSO. Si lo comparamos con los GA, POS es más fácil de implementar y hay pocos parámetros que ajustar.

Ant Colony Optimization (ACO)

Este método propuesto por Dorigo M. y otros (1996) es un ejemplo, como el Temple Simulado, Redes Neuronales y otros, del afortunado uso de metáforas naturales para diseñar un algoritmo de optimización. En este caso se aprecia con claridad como las soluciones generadas previamente afectan a las soluciones que se generan en el futuro.

Las hormigas reales son capaces de encontrar el camino más corto desde una fuente de comida al hormiguero sin usar señales visuales. También son capaces de adaptarse a cambios del entorno, por ejemplo, encontrando un nuevo camino más corto cuando el anterior ya no es factible debido a un obstáculo interpuesto. Las hormigas se mueven en línea recta que conecta la fuente de comida con su hormiguero. El medio básico que tienen para formar y mantener la línea es un reguero de pheromone; al caminar depositan determinada cantidad de esta sustancia y cada hormiga prefiere (probabilísticamente) seguir una dirección rica en pheromone. Cuando aparece de forma imprevista un obstáculo no esperado que interrumpe el camino inicial, aquellas hormigas que están justo en frente del obstáculo no pueden continuar siguiendo el reguero de pheromone: tienen que elegir entre girar

a la izquierda o a la derecha. En esta situación podemos esperar que la mitad de las hormigas elijan una cosa y la otra mitad otra. Una situación muy similar puede encontrarse en el otro lado del obstáculo. Es interesante destacar que aquellas hormigas que eligen por casualidad el camino corto, reconstruyen más rápidamente el reguero de pheromone interrumpido, en comparación con aquellas que eligen el camino largo. Así que el camino corto recibirá más cantidad de pheromone por unidad de tiempo y en cada turno un mayor número de hormigas elegirá el camino corto. Debido a este proceso retroalimentado positivo, todas las hormigas elegirán rápidamente el camino más corto.

Dorigo M. y Gambardella L.M. (1997) han aplicado esta técnica al TSP basándose en las ideas que se comentan a continuación. Una hormiga artificial es un agente que se mueve de una ciudad a otra en un grafo de TSP. Elige la ciudad a la que moverse (o arco que añade a su ruta) con una probabilidad proporcional al reguero acumulado y la distancia del arco que se añade. Las hormigas artificiales prefieren probabilísticamente ciudades que están conectadas por arcos con mucho reguero de pheromone y que están próximas. Inicialmente, m hormigas artificiales se colocan en ciudades seleccionadas aleatoriamente. En cada iteración se mueven a nuevas ciudades y modifican el reguero de pheromone de los arcos usados - esto se denomina *actualización de reguero local.* Cuando todas las hormigas han completado una ruta, la hormiga que hace la ruta más corta modifica los arcos pertenecientes a su ruta - se denomina *actualización de reguero global*- añadiendo una cantidad de reguero de pheromone que es inversamente proporcional a la longitud de la ruta.

Hay tres ideas de la conducta de las hormigas que se transfieren a la colonia de hormigas artificiales:

1. la preferencia por caminos con alto nivel de pheromone,
2. el alto ratio de crecimiento de la cantidad de pheromone en los caminos cortos, y
3. el reguero mediador de comunicación entre las hormigas.

Se ha dado a las hormigas artificiales algunas capacidades que no tienen sus colegas naturales pero que se ha observado que son aptas para su aplicación al TSP: las hormigas artificiales pueden determinar la distancia a la que están las ciudades y están dotadas con una memoria de trabajo M_k usada para memorizar las ciudades ya visitadas (la memoria de trabajo está vacía al comienzo de cada nueva ruta y se actualiza después de cada paso de tiempo añadiendo las nuevas ciudades visitadas).

Con el fin de evitar que un arco muy atractivo sea elegido por todas las hormigas se realiza la actualización local del reguero: cada vez que un arco es elegido por una hormiga su cantidad de pheromone se cambia aplicando la fórmula de actualización local. La evaporación del reguero de pheromone en el mundo de

las hormigas reales se traslada a la colonia de hormigas artificiales en forma de actualización local del reguero.

3.3 Métodos basados en muestreo aleatorio

Simulated Annealing (SA)

Kirkpatrick S. y otros (1983), e independientemente Cerny V. (1985), proponen un procedimiento para obtener soluciones aproximadas llamado Simulated Annealing (Temple Simulado o Recocido Simulado). Se pueden considerar como una variante de los métodos de búsqueda local, en la que se permite empeoramientos en la solución actual aunque de forma controlada.

Los autores mencionados introdujeron el concepto de *templado* en optimización combinatoria. Este concepto está basado en una estrecha analogía entre el proceso físico de *templado* y los problemas combinatorios. La idea original que dio lugar a esta metaheurística es el denominado *"algoritmo de Metrópolis",* Metrópolis y otros (1953), bien conocido en el mundo de la Química-Física. Para estudiar las propiedades de equilibrio, Metrópolis utilizó el *"método de Montecarlo",* que es el más usado en Mecánica Estadística para estudiar el comportamiento microscópico de los cuerpos.

Las moléculas de una sustancia pueden tener distintos niveles de energía. El menor de estos niveles es el llamado *"estado fundamental",* γ_0. A una temperatura de 0º K todas las moléculas están en su estado fundamental, pero se sabe que un trozo de sustancia a alta temperatura probablemente posea un estado de energía más alto que otro idéntico a temperatura menor.

Cada una de las maneras en que las moléculas pueden estar distribuidas entre los distintos niveles de energía recibe el nombre de *microestado.* Se denomina Ω al conjunto de todos los posibles microestados y *número de ocupación,* n_i, al número de partículas en el nivel de energía i. El número de moléculas en los estados superiores decrece para una temperatura T fija.

Para reducir la energía de la sustancia al menor valor posible, bajar simplemente la temperatura al cero absoluto no asegura necesariamente que la sustancia alcance su configuración energética más baja posible. En *física termodinámica,* se conoce como *templado* a un proceso termal para obtener los estados de más baja energía de un sólido en un recipiente. Para ello hay que elevar la temperatura del recipiente, al menos hasta conseguir que el sólido se funda y posteriormente bajar la temperatura del recipiente muy suavemente hasta que las partículas se estabilicen, es decir, hasta llegar al estado sólido; entonces se dice que se ha producido la *congelación.*

Simulación Termodinámica Optimización Combinatoria Estados del material Soluciones factibles S Energía Función f Estados surgidos por mecanismo de perturbación Soluciones vecinas Estados metaestables Mínimo local Estado de congelación Solución final

Simulación Termodinámica	**Optimización Combinatoria**
Estados del material	Soluciones factibles S
Energía	Función f
Estados surgidos por mecanismo de perturbación	Soluciones vecinas
Estados metaestables	Mínimo local
Estado de congelación	Solución final

Fig. 8 Correspondencia entre los elementos de simulación termodinámica y optimización combinatoria.

Durante la fase líquida, todas las partículas del material se mueven de forma aleatoria. Cuando se llega al estado sólido, las partículas están ordenadas en una estructura *enrejada* con energía mínima. Esta estructura se consigue solamente si la temperatura inicial es suficientemente alta, y el enfriamiento se hace de forma suficientemente lenta; de lo contrario, el material alcanza una estructura metaestable con mayor valor energético. La correspondencia entre los elementos de simulación termodinámica y la optimización combinatoria es la que aparece en la figura 8.

GRASP

GRASP son las iniciales en inglés de Greedy Randomize Adaptive Search Procedures (Procedimientos de Búsqueda basados en funciones Avidas, Aleatorias y Adaptativas); se dieron a conocer a finales de los ochenta en el trabajo de Feo T.A. y Resende M.G.C. (1989), pero han tenido un desarrollo más reciente que los otros metaheurísticos. Una amplia descripción se puede encontrar en un trabajo posterior de los mismos autores Feo T.A. y Resende M.G.C. (1995).

GRASP es una técnica simple aleatoria e iterativa, en la que cada iteración provee una solución al problema que se esté tratando. La mejor solución de todas las iteraciones GRASP se guarda como resultado final. Hay dos fases en cada iteración GRASP: la primera construye secuencial e inteligentemente una solución inicial por medio de una función ávida, aleatoria y adaptativa; en la segunda se fase aplica un procedimiento de búsqueda local a la solución construida, con la esperanza de encontrar una mejora.

En la fase de construcción se va añadiendo en cada paso un elemento, hasta obtener la solución completa. En cada iteración, la elección del próximo elemento para ser añadido a la solución parcial, viene determinado por una función *ávida (greedy).* Esta función mide el beneficio, según la función objetivo, de añadir cada

uno de los elementos y elige la mejor opción. Esta medida es *miope,* en el sentido de que no tiene en cuenta qué ocurrirá en iteraciones sucesivas al realizar una elección, sino únicamente lo que pasa en esa iteración.

El heurístico es *adaptativo,* ya que los beneficios asociados con cada elemento se actualizan en cada iteración de la fase de construcción, para reflejar los cambios producidos por la selección de elementos previos.

GRASP es *aleatorizado* porque no selecciona el mejor candidato según la función ávida adaptada. Con los mejores elementos a añadir se construye una lista denominada *Lista Restringida de Candidatos* (RCL en inglés), y se elige de forma aleatoria uno de los mejores candidatos de dicha lista, que no será necesariamente el mejor. La aleatoriedad sirve como mecanismo de diversificación en GRASP.

No se garantiza que la solución generada por la fase de construcción de GRASP sea un óptimo local respecto a una definición simple de vecindario. Por ello se aplica búsqueda local para mejorar cada solución construida. La fase de búsqueda local finaliza cuando no se encuentra una solución mejor en el vecindario de la solución actual. GRASP se basa en realizar múltiples iteraciones y quedarse con la mejor, por lo que no es especialmente beneficioso para el método el detenerse demasiado en mejorar una solución dada. El éxito de esta segunda fase viene determinado por la acertada elección de la estructura del vecindario, técnicas eficientes de búsqueda en vecindarios, y la solución inicial.

GRASP, al igual que otros metaheurísticos, se ha combinado también con Path Relinking, estrategia comentada en el punto 3.3.2. La idea es encadenar las soluciones que se obtienen al final de la segunda fase, utilizando una para iniciar la búsqueda y otra como guía.

Cross-Entropy

Los precedentes de CE los encontramos en Rubinstein R. (1997). CE es un método para resolver problemas de optimización combinatoria, optimización continua con múltiples extremos y simulación de eventos poco frecuentes. Se basa en la idea de transformar el problema de optimización determinista original en un problema estocástico asociado y afrontar dicho problema asociado utilizando un algoritmo adaptativo. Se construye una serie aleatoria de soluciones que convergen probabilísticamente al óptimo o cerca del óptimo. Tras definir el problema estocástico asociado CE emplea dos fases: 1. Generación de un conjunto de datos aleatorios (trayectorias, vectores,..) según un mecanismo aleatorio específico. 2. Actualización de los parámetros del mecanismo aleatorio, en base a los datos para producir una mejor muestra en la siguiente iteración.

La importancia de este método es que define una meticulosa estructura matemática para obtener rápidamente normas de aprendizaje y actualización en cierta forma "óptimas" basadas en teoría de simulación avanzada. Hay que re-

saltar que CE puede ser aplicado satisfactoriamente tanto a problemas determinísticos como aleatorios.

4 Bibliografía

[1] Campos V., F. Glover, M. Laguna, R. Martí. An Experimental Evaluation of a Scatter Search for the Linear Ordering Problem. Journal of Global Optimization. Vol. 21, nº 4, pp. 397-414, 2001.

[2] Cerny V., Thermodynamical approach to the Traveling Salesman Problem: An efficient simulation algorithm. Journal of Optimization Theory and Applications, 45, pp. 41-51, 1985.

[3] Charnes A., W.W. Cooper . Management Models and Industrial Applications of Linear Programming. Wiley, New York. 1961.

[4] Cochrane J., M. Zeleny (eds), Multiple Criteria Decision Making. University of South Carolina Press. 1973.

[5] Crowston W.B., F. Glover, G.L. Thompson, J.D. Trawick. Probabilistic and Parametric Learning Combinations of Local Job Shop Scheduling Rules. ONR Research Memorandum No. 117, GSIA, Carnegie Mellon University. Pittsburgh, PA. 1963.

[6] De Boer PT, Dirk P. Kroese, Shie Mannor, Reuven Y. Rubinstein. A Tutorial on the Cross-Entropy Method. Annals of Operations Research. Vol 134, pp. 19-67. 2005.

[7] Dorigo M., L.M. Gambardella. Ant Colonies for the Traveling Salesman Problems. TR/IRIDIA/1.996-3. Université Libre de Bruxelles. Belgium. Publicado en BioSystem, Vol 43, pp.73-81. 1997.

[8] Dorigo M., V. Meniezzo, A. Colorni (1996). The Ant System: Optimization by a Colony of Cooperating. Agents. IEEE Transactions on Systems, Man, and Cybernetics-Part B, 26 (1):29-41.

[9] Feo T. A., M. G. C. Resende (1989). A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. Operations Research Letters. Vol. 8, pp. 6771.

[10] Feo T. A., M. G. C. Resende (1995). Greedy Randomized Adaptive Search Procedures. Journal of Global Optimization. Vol. 2, pp. 127.

[11] Fisher H ., G. L. Thompson (1963). Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules. Industrial Scheduling, J.F. Muth and G. L. Thompson (eds.) Prentice-Hall, pp. 225-251.

[12] Fogel L. J., A. J. Owens, M. J. Walsh (1966). Artificial Intelligence Through Simulated Evolution. Wiley, New York.

[13] Garey M.R., D. Johnson (1979). Computers and Intractability. A guide to the Theory of NP-Completeness. W.H Freeman and Company, New York.

[14] Garfinkel R.S. (1985). Motivation and Modelling in LAWLER E.L., LESTRA J.K., RINNOOY KAN A.H.G. & SHMOYS D.B. (eds.) The Travelling Salesman Problem: A Guide Tour of Combinatorial Optimization. Wiley. Chichester, pp. 17-36.

[15] Glover F. (1989). Tabu Search: Part I. ORSA Journal on Computing. Vol. 1, pp. 190206.

[16] Glover F. (1990-a). Tabu Search: Part II. ORSA Journal on Computing. Vol. 2, pp. 432.

[17] Glover F. (1996). Búsqueda Tabú en Optimización Heurística y Redes Neuronales. Adenso Díaz (coordinador). Paraninfo. Madrid. pp. 105143.

[18] Glover F. (1998). A Template for Scatter Search and Path Relinking. J.k. Hao, E. Lutton, E. Ronald, M. Schoenauer, D. Syners (eds.). artificial Evolution. Springer LNCS 1363, pp. 13-64.

[19] Glover, F., M. Laguna (1997). Tabu Search. Kluwer Academic Publishers. Boston.

[20] Glover F., M. Laguna (2002). Tabu Search. Handbook of Applied Optimization, P. M. Paradalos and M. G. S. Resende (eds). Oxford University Press, pp. 194-208.

[21] Glover F., M. Laguna, R. MARTI (2004). New Ideas and Applications of Scatter Search and Path Relinking. Techniques in Engineering, G. C. Onwubolu and B. V. Babu (eds.), Springer, ISBN: 3-540-20167-X .

[22] Glover F., M. Laguna, R. MARTI (2000). Fundamentals of Scatter Search and Path Relinking. Control and Cybernetics, vol. 39, no. 3 pp. 653-684

[23] Goldberg D.E. (1994). Genetic and Evolutionary Algorithms Come of Age. Communications of the ACM. Vol. 37, No. 3. March.

[24] Hansen P., N. Mladenović (1998). An Introduction to Variable Neighborhood Search. S. Voss et al. (eds). Metaheuristics Advances and Trends in Local Search Paradigms for Optimization, pp. 433-458, MIC-97, Kluwer, Dordrecht.

[25] Hansen P., N. Mladenović (2003). A Tutorial on Variable Neighborhood Search. Technical report. Les Cahiers du GERAD G-2003-46. Montréal. HOLLAND J. (1975). Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Abor.

[26] Kennedy J., R. Eberhart (1995). Particle swarm optimization, in Proc. of the IEEE Int. Conf. on Neural Networks, Piscataway, NJ, pp. 1942–1948,

[27] Kirkpatrick S., JR. C. D. Gelatt, M. P Vecchi. (1983). Optimization by Simulated Annealing. Science, Vol. 220, pp. 671-680.

[28] Laguna M. (2002). Scatter Search. In Handbook of Applied Optimization, P. M. Pardalos and M. G. C. Resende (Eds.), Oxford University Press, pp. 183-193.

[29] Lawler E. (1976). Combinatorial Optimization. Networks and Matroids, Holt, Rinehart and Winston, New York.

[30] Lawler E. L., J. K. Lenstra, A. H. G. Rinnooy Kan, D. B. Shmoys (eds.) (1985). The Traveling Salesman Problem: A Guide Tour of Combinatorial Optimization. Wiley. Chichester.

[31] Metropolis N., A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller (1953). Equation of State Calculations by Fast Computing Machines. Journal of Chemical Physics, 21, pp. 1087–1092.

[32] Mladenović N. (1995). A Variable Neighborhood Algorithm – A New Metaheuristic for Combinatorial Optimization. Abstracts of papers presented at Optimization Days. Montreal, p. 112.

[33] Mladenović N., P. Hansen (1997). Variable Neighborhood Search. Computers and Operations Research, 24, 1097-1100.

[34] Naylor T.H., J. Balintfy, D. Burdick, K. Chu (1982). Técnicas de Simulación en Computadoras. Limusa Méjico.

[35] Osman I. H., J. P. Kelly (1996). Meta-Heuristics: Theory & Applications. Kluwer Academic Publishers.

[36] Papadomitriou C. H., K. Steiglitz (1982). Combinatorial Optimization: Algorithms and Complexity. PrenticeHall, New York.

[37] Pareto, V. (1896). Cours D'Economie Politique, Vol. I y II. F. Rouge, Lausanne.

[38] Resende M. G. C., C. C. Ribeiro (2005). GRASP with Path- Relinking: Recent Advances and Applications. Metaheuristics: Progress as Real Problem Solvers, T. Ibaraki, K. Nonobe and M. Yagiura, (Eds.), Springer, pp.29-63

[39] Rubinstein R. (1997). Optimization of computer simulation models with rare events. European Journal of Operations Research, 99, 89-112.

[40] Schrijver A. (1986). Theory of Linear and Integer Programming. Series in Discrete Mathematics. John Wiley & Sons.

Introducción a la Búsqueda Tabú*

Belén Melián Batista[a], **Fred Glover**[b]

[a] Dpto. de Estadística, I.O. y Computación,
Escuela Técnica Superior de Ingeniería Informática
Universidad de La Laguna
mbmelian@ull.es

[b] University of Colorado at Boulder,
Fred.Glover@Colorado.edu

1 Introducción

El término Búsqueda Tabú (*Tabu Search* - TS) fue introducido en 1986 por Fred Glover en el mismo artículo que introdujo el término metaheurística [5]. Los principios fundamentales de la búsqueda fueron elaborados en una serie de artículos de finales de los años 80 y principios de los 90, que fueron luego unificados en el libro "Tabu Search" en 1997 [8]. El destacado éxito de la búsqueda tabú para resolver problemas de optimización duros, especialmente aquellos que surgen en aplicaciones del mundo real, ha causado una explosión de nuevas aplicaciones durante los últimos años, que aparecen resumidas en [9].

La búsqueda tabú es una metaheurística que guía un procedimiento heurístico de búsqueda local en la búsqueda de optimalidad global. Su filosofía se basa en derivar y explotar una colección de estrategias inteligentes para la resolución de problemas, basadas en procedimientos implícitos y explícitos de aprendizaje. El marco de memoria adaptativa de la búsqueda tabú no sólo explota la historia del proceso de resolución del problema, sino que también exige la creación de

*Este trabajo ha sido parcialmente financiado por el Ministerio de Ciencia y Tecnología (proyecto TIN2005-08404-C04-03 (70% son fondos FEDER)) y por el Gobierno de Canarias (proyecto PI042004/088). La actividad desarrollada se enmarca dentro de los objetivos de la red RedHeur (proyectoTIN2004-20061-E).

estructuras para hacer posible tal explotación. De esta forma, los elementos prohibidos en la búsqueda tabú reciben este estatus por la confianza en una memoria evolutiva, que permite alterar este estado en función del tiempo y las circunstancias. En este sentido es posible asumir que la búsqueda tabú está basada en determinados conceptos que unen los campos de inteligencia artificial y optimización.

Más particularmente, la búsqueda tabú está basada en la premisa de que para clasificar un procedimiento de resolución como inteligente, es necesario que éste incorpore *memoria adaptativa* y *exploración responsiva*. La memoria adaptativa en búsqueda tabú permite la implementación de procedimientos capaces de realizar la búsqueda en el espacio de soluciones eficaz y eficientemente. Dado que las decisiones locales están por tanto guiadas por información obtenida a lo largo del proceso de búsqueda, la búsqueda tabú contrasta con diseños que por contra confían en procesos semialeatorios, que implementan una forma de muestreo. La memoria adaptativa también contrasta con los típicos diseños de memoria rígidos tales como las estrategias de ramificación y acotación.

El énfasis en la exploración responsiva considerada en la búsqueda tabú deriva de la suposición de que una mala elección estratégica puede proporcionar más información que una buena elección realizada al azar, dado que una elección estratégica mala puede proporcionar pistas útiles sobre cómo guiar la búsqueda hacia zonas prometedoras. Por lo tanto, la exploración responsiva integra los principios básicos de la búsqueda inteligente; explota las características de las soluciones buenas a la vez que explora nuevas regiones prometedoras.

2 La estructura de la Búsqueda Tabú

2.1 Uso de memoria

Las estructuras de memoria de la búsqueda tabú funcionan mediante referencia a cuatro dimensiones principales, consistentes en la propiedad de ser reciente, en frecuencia, en calidad, y en influencia. Las memorias basadas en lo reciente y en frecuencia se complementan la una a la otra para lograr el balance entre intensificación y diversificación que todo proceso de búsqueda heurística debe poseer. Discutiremos con más detalle los aspectos referentes a estas dos primeras dimensiones de memoria a lo largo de este capítulo. La dimensión de *calidad* hace referencia a la habilidad para diferenciar la bondad de las soluciones visitadas a lo largo del proceso de búsqueda. De esta forma, la memoria puede ser utilizada para la identificación de elementos comunes a soluciones buenas o a ciertos caminos que conducen a ellas. La calidad constituye un fundamento para el aprendizaje basado en incentivos, donde se refuerzan las acciones que conducen a buenas soluciones y se penalizan aquellas que, por contra, conducen a soluciones pobres. La flexibilidad de las estructuras de memoria mencionadas hasta el momento

permiten guiar la búsqueda en un entorno multi-objetivo, dado que se determina la bondad de una dirección de búsqueda particular mediante más de una función. Por último, la cuarta dimensión de memoria, referida a la *influencia*, considera el impacto de las decisiones tomadas durante la búsqueda, no sólo en lo referente a la calidad de las soluciones, sino también en lo referente a la estructura de las mismas. Este último uso de memoria es una característica importante de la búsqueda tabú que con frecuencia se olvida, pero que debería ser considerada incluso en los diseños más simples como veremos a lo largo de este capítulo.

El uso de memoria en la búsqueda tabú es tanto *explícita* como *implícita*. En el primer caso, se almacenan en memoria soluciones completas, generalmente soluciones élite visitadas durante la búsqueda, mientras que en el segundo caso, se almacena información sobre determinados atributos de las soluciones que cambian al pasar de una solución a otra. Aunque, en algunos casos, la memoria explícita es usada para evitar visitar soluciones más de una vez, esta aplicación es limitada dado que es necesario la implementación de estructuras de memoria muy eficientes para evitar requerimientos de memoria excesivos. De cualquier manera, estos dos tipos de memoria son complementarios, puesto que la memoria explícita permite expandir los entornos de búsqueda usados durante un proceso de búsqueda local mediante la inclusión de soluciones élite, mientras que la memoria basada en atributos los reduce prohibiendo determinados movimientos.

2.2 Intensificación y Diversificación

Las estrategias de intensificación y diversificación constituyen dos elementos altamente importantes en un proceso de búsqueda tabú. Las estrategias de intensificación se basan en la modificación de reglas de selección para favorecer la elección de buenas combinaciones de movimientos y caraterísticas de soluciones encontradas. Esto implica que es necesario identificar un conjunto de soluciones élite cuyos buenos atributos puedan ser incorporados a nuevas soluciones creadas. La pertenencia al conjunto de soluciones élite se determina generalmente atendiendo a los valores de la función objetivo comparados con la mejor solución obtenida hasta el momento.

Por otro lado, las estrategias de diversificación tratan de conducir la búsqueda a zonas del espacio de soluciones no visitadas anteriormente y generar nuevas soluciones que difieran significativamente de las ya evaluadas.

2.3 Un ejemplo ilustrativo

Los problemas de permutaciones son una clase importante de problemas en optimización, y ofrecen un modo muy útil para demostrar algunas de las consideraciones que deben ser tratadas en el dominio combinatorio. Las instancias clásicas de problemas de permutaciones incluyen los problemas del viajante de

comercio, asignación cuadrática, secuenciación de la producción, y una variedad de problemas de diseño. Como base para la ilustración, consideremos el problema de secuenciación de tareas en una única máquina. El objetivo de este problema es encontrar un orden para secuenciar las tareas en la máquina de tal forma que se minimice el retraso total en la ejecución de las tareas. Cada tarea $j, j = 1, 2, ..., n$, tiene asignados un tiempo de procesamiento p_j y un día de finalización d_j. De forma adicional, se podría considerar un valor de penalización por retardo en la finalización de las tareas que dependería de la tarea considerada, w_j. Por tanto, la función a minimizar se expresa como

$$F = \sum_{j=1}^{n} w_j [C_j - d_j]^+,$$

donde C_j es el tiempo de finalización de la tarea j y $[C_j - d_j]^+ = \max\{0, C_j - d_j\}$. El tiempo de finalización de una tarea j, C_j, es igual al tiempo de procesamiento de la tarea j más la suma de los tiempos de procesamiento de todas las tareas que se realizan antes que j.

El problema consiste en determinar el orden de secuenciación de las tareas que minimiza el valor de la función objetivo F. Una secuenciación de las tareas, que constituye una permutación, define completamente a una solución.

Nos centramos, por tanto, en el problema de secuenciación de tareas en una única máquina para introducir e ilustrar los componentes básicos de la búsqueda tabú. Supongamos que se consideran 6 tareas para su secuenciación en la máquina. A modo de ilustración, supongamos que este problema de 6 tareas tiene tiempos de procesamiento dados por (5, 8, 2, 6, 10, 3), días de terminación especificados por (9, 10, 16, 7, 20, 23), y penalizaciones por retraso $w_j = 1$ para $j = 1, 2, ..., 6$. Deseamos diseñar un método capaz de encontrar una solución óptima o cercana a la óptima explorando sólo un pequeño subconjunto de todas las permutaciones posibles.

Primero asumimos que puede construirse una solución inicial para este problema de alguna manera inteligente, es decir, tomando ventaja de la estructura específica del problema. Supongamos que la solución inicial de nuestro problema es la que aparece en la Figura 1.

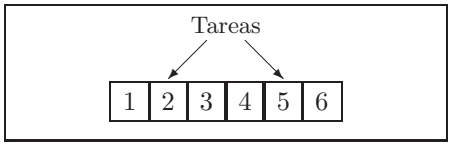


Figura 1: Permutación inicial

La ordenación de la Figura 1 especifica que la tarea 1 se realiza en primer

Tareas i	j	F	valor del movimiento	$abs(d_i - d_j)$
1	2	40	1	9
1	3	42	3	7
1	4	38	-1	2
1	5	64	25	11
1	6	47	8	14
2	3	41	2	6
2	4	37	-2	3
2	5	49	10	10
2	6	39	0	13
3	4	42	3	9
3	5	54	15	4
3	6	48	9	7
4	5	43	4	13
4	6	38	-1	16
5	6	32	-7	3

Tabla 1: Entorno de Intercambios

lugar, seguida por la tarea 2, etc. El valor de la función objetivo para esta solución es 39. Los métodos TS operan bajo el supuesto de que se puede construir un entorno para identificar "soluciones adyacentes" que puedan ser alcanzadas desde la solución actual. Los intercambios por pares son frecuentemente usados para definir entornos en problemas de permutaciones, identificando movimientos que conducen una solución a la siguiente. En nuestro problema, un intercambio cambia la posición de dos tareas como se ilustra en la Figura 2. Por tanto, el entorno completo de una solución en nuestro ejemplo ilustrativo está formado por 15 soluciones adyacentes que pueden ser obtenidas a partir de estos intercambios tal como muestra el Cuadro 1.

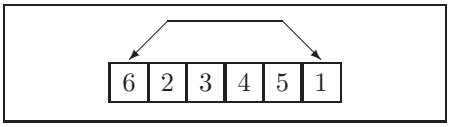


Figura 2: Intercambio de las tareas 1 y 6

Tal como observamos en el Cuadro 1, asociado a cada intercambio hay un valor de movimiento, que representa el cambio sobre el valor de la función objetivo como resultado del intercambio realizado. Los valores de los movimientos generalmente proporcionan una base fundamental para evaluar la calidad de los mismos, aunque también pueden ser importantes otros criterios. Un mecanismo principal para explotar la memoria en la búsqueda tabú es clasificar un subcon-

junto de movimientos en un entorno como prohibidos (o tabú). La clasificación depende de la historia de la búsqueda, determinada mediante lo reciente o frecuente que ciertos movimientos o componentes de soluciones, llamados *atributos*, han participado en la generación de soluciones pasadas. Por ejemplo, un atributo de un movimiento es la identidad del par de elementos que cambian posiciones (en este caso, las dos tareas intercambiadas). Como base para evitar la búsqueda desde combinaciones de intercambio repetidas usadas en el pasado reciente, invirtiendo potencialmente los efectos de movimientos anteriores por intercambios que podrían devolver a posiciones previas, clasificaremos como tabú todos los intercambios compuestos por cualquiera de los pares de tareas más recientes; en este caso, para propósitos ilustrativos, las tres más recientes. Esto significa que un par de tareas será tabú durante un período de 3 iteraciones. Dado que intercambiar las tareas 2 y 5 es lo mismo que intercambiar las tareas 5 y 2, ambos intercambios pueden ser representados por el par $(2, 5)$. Por lo tanto, se puede usar una estructura de datos como la usada en la Figura 3.

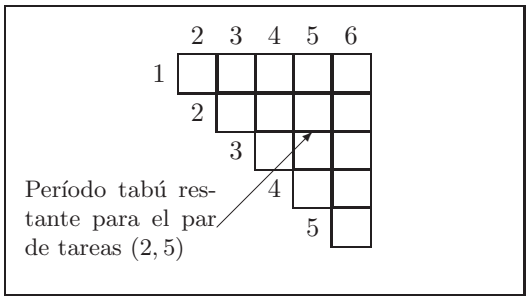


Figura 3: Estructura de Datos Tabú

Cada celda de la estructura de la Figura 3 contiene el número de iteraciones restantes hasta que las capas correspondientes puedan nuevamente intercambiar posiciones. Por tanto, si la celda $(2, 5)$ tuviera un valor de cero, entonces las tareas 2 y 5 estarían disponibles para intercambiar posiciones. Por otro lado, si la celda tuviera un valor de 2, entonces las tareas no podrían intercambiar posiciones durante las dos iteraciones siguientes (es decir, un intercambio que cambia estas tareas es clasificado como tabú).

Para implementar restricciones tabú, debe tenerse en cuenta una excepción importante: las restricciones tabú no son inviolables bajo cualquier circunstancia. Cuando un movimiento tabú resultara en una solución mejor que cualquiera visitada hasta ese momento, su clasificación tabú podría ser reemplazada. Una condición que permite que ocurra tal reemplazo se llama *criterio de aspiración*. A continuación se muestran 7 iteraciones del procedimiento de búsqueda tabú básico, que usa la restricción tabú de tareas emparejadas.

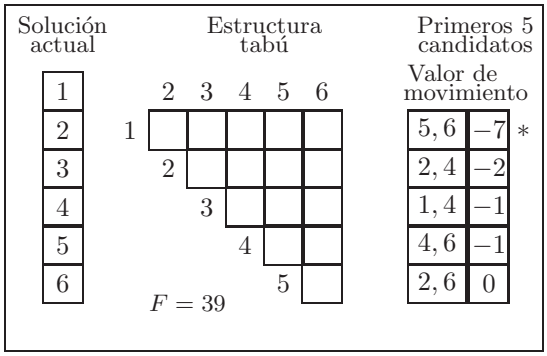


Figura 4: Iteración 0

La solución de partida de la Figura 4 tiene un valor de la función objetivo $F = 39$, y la estructura de los datos tabú está inicialmente vacía, es decir, está llena de ceros, indicando que ningún movimiento está clasificado como tabú al comienzo de la búsqueda. Después de evaluar los movimientos de intercambio de candidatos, se muestran en la tabla para la iteración 0 los cinco primeros movimientos (en términos de valores de movimiento). Para minimizar localmente el retraso total en la ejecución de las tareas, intercambiamos las posiciones de las tareas 5 y 6, como se indica a través del asterisco en la Figura 4. El decremento total de este movimiento es igual a 7 unidades, con lo que el valor de la función objetivo pasa a ser $F = 32$.

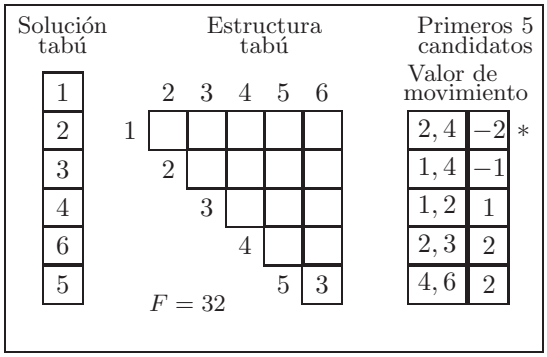


Figura 5: Iteración 1

La nueva solución actual tiene un valor de función objetivo $F = 32$ (es decir,

el retraso total anterior más el valor del movimiento seleccionado). La estructura tabú de la Figura 5 ahora muestra que el intercambio de las posiciones de las tareas 5 y 6 se prohíbe durante 3 iteraciones. El movimiento que proporciona la mayor mejora en este paso es el intercambio de las tareas 2 y 4 con un decremento de 2 unidades.

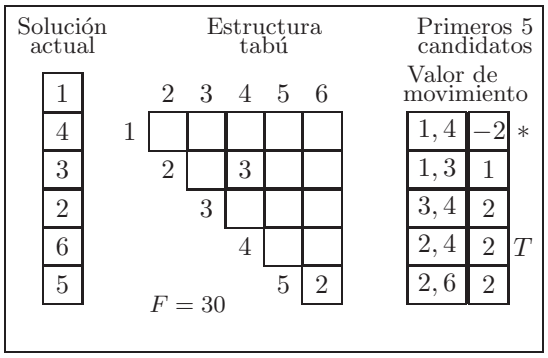


Figura 6: Iteración 2

La nueva solución actual tiene un retraso en la ejecución de las tareas de 30. En esta iteración se clasifican como tabú dos intercambios como se indica mediante las entradas distintas de cero en la estructura tabú de la Figura 6. Note que la entrada $(5, 6)$ ha disminuido de 3 a 2, indicando que su período tabú original de 3 ahora tiene 2 iteraciones restantes. En este momento, el intercambio de las tareas 1 y 4 conduce a una nueva mejora en el valor de la función objetivo, disminuyendo en dos unidades. La Figura 7 muestra ahora 3 movimientos clasificados como tabú.

En este momento, ninguno de los candidatos tiene un valor de movimiento negativo. Por lo tanto, se realiza un movimiento de no mejora. Dado que el primer movimiento de no mejora es el inverso del movimiento ejecutado en la iteración anterior, que está clasificado como tabú (indicado por T), este movimiento no se selecciona. Entonces se elige el intercambio de las tareas 1 y 3, como se indica en la Figura 7.

Siguiendo el mismo procedimiento indicado hasta este momento, se realizarían las siguientes iteraciones mostradas en las Figuras 8 a 11. En estas últimas iteraciones observamos que se realizan movimientos de no mejora para escapar de la solución con valor objetivo $F = 28$, que parece ser un óptimo local.

Si durante el proceso explicado hubiera habido algún movimiento clasificado como tabú que condujera a una solución con un retraso en la finalización de las tareas menor que el de la mejor solución encontrada hasta el momento, se podría haber usado un criterio de aspiración. En este caso, hubiéramos usado el criterio

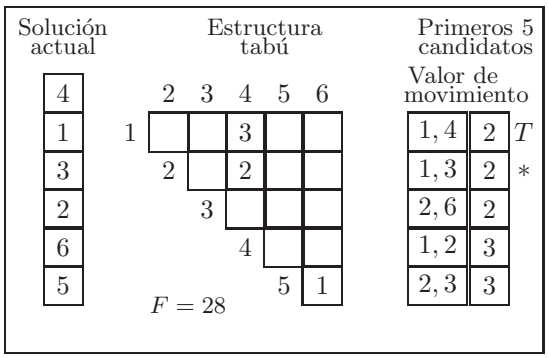


Figura 7: Iteración 3

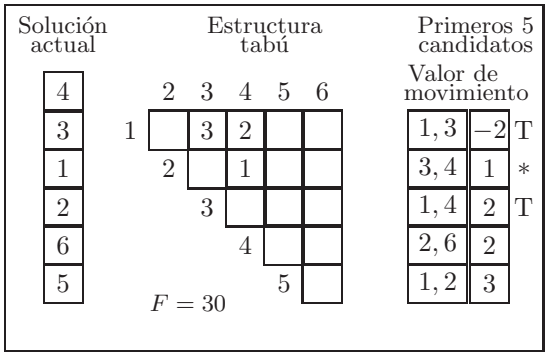


Figura 8: Iteración 4

de aspiración por objetivo, que selecciona como solución actual la que tenga un menor valor objetivo, independientemente de que los movimientos requeridos para alcanzarla sean tabú.

En algunas situaciones, puede ser deseable incrementar el porcentaje de movimientos disponibles que reciben una clasificación tabú. Además, a pesar del tipo de restricción seleccionado, a menudo se obtienen mejores resultados por los plazos tabú que varían dinámicamente, como se describe con posterioridad en este capítulo.

Valores de Movimiento y Estrategia de Lista de Candidatos. Dado que la búsqueda tabú selecciona agresivamente los mejores movimientos admisibles (donde el significado de mejor es afectado por la clasificación tabú y otros elementos a ser indicados), debe examinar y comparar un número de opciones de movimiento.

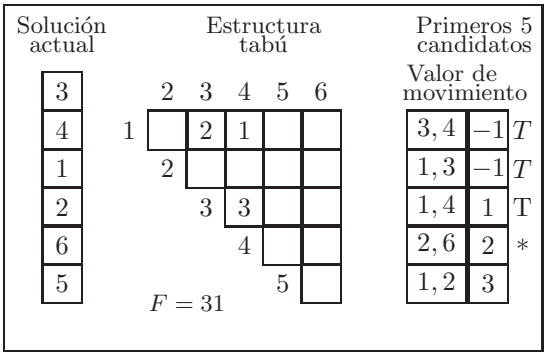


Figura 9: Iteración 5

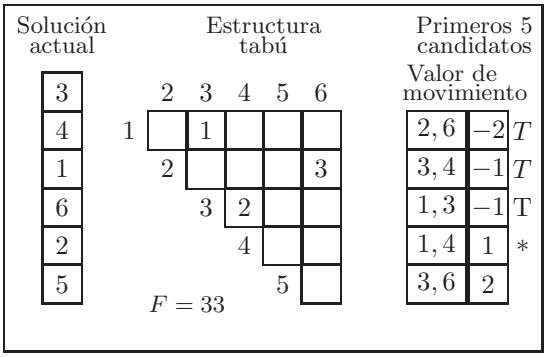


Figura 10: Iteración 6

Para muchos problemas, sólo una porción de los valores de movimiento cambia de una iteración a otra, y a menudo estos valores cambiados pueden ser separados y actualizados muy rápidamente. Este elemento de mantener actualizaciones eficientes es muy importante y en ocasiones ignorado. Por ejemplo, en la presente ilustración puede ser útil almacenar una tabla *valor_movimiento(j,k)*, que almacena el actual valor del movimiento para intercambiar las tareas j y k. Entonces cuando se ejecuta un movimiento, una parte relativamente pequeña de esta tabla (formada por los valores que cambian) puede ser modificada rápidamente, y la tabla actualizada puede ser consultada para identificar movimientos que se convierten en los nuevos candidatos superiores.

Si atendemos al Cuadro 1, observamos claramente que hay una variación muy grande en la calidad de cada intercambio en el entorno definido para una solución.

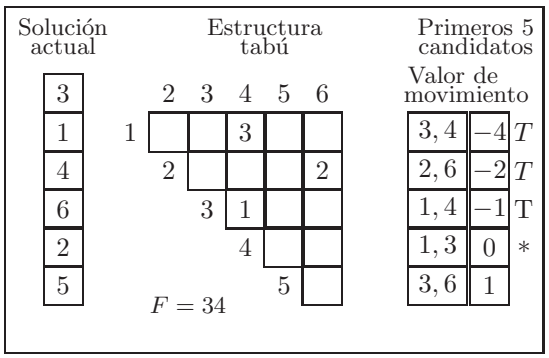


Figura 11: Iteración 7

Por lo tanto, parece útil eliminar algunas soluciones de calidad baja antes de evaluar su valor de movimiento mediante un filtro. En nuestro ejemplo ilustrativo de secuenciación de tareas en una máquina, podemos implementar una regla que elimine aquellos movimientos para los que el valor absoluto de la diferencia de los días de terminación sea mayor que 3. De esta forma, en los datos del Cuadro 1, evaluaríamos tan sólo 3 movimientos en vez de 15, generando así una lista de candidatos.

Estructuras de Memoria Tabú Complementarias.

El complemento de *memoria basada en lo reciente* a la *memoria basada en frecuencia* añade una componente que típicamente opera sobre un horizonte más largo. En nuestro ejemplo ilustrativo, si continuamos con la traza anterior utilizando únicamente información basada en las 3 iteraciones más recientes, observamos que se produce un ciclado de las soluciones. Para ilustrar una de las aplicaciones útiles de largo período de memoria basada en frecuencia, suponemos que han sido ejecutadas 14 iteraciones TS, y que el número de veces que cada par de tareas ha sido intercambiado se guarda en una estructura de datos tabú expandida (Figura 12). La diagonal inferior de esta estructura ahora contiene los contadores de frecuencia.

En la iteración actual (iteración 15), la memoria basada en lo reciente indica que los últimos tres pares de tareas intercambiados fueron $(1, 4)$, $(2, 6)$, y $(3, 4)$. Los contadores de frecuencia muestran la distribución de movimientos a través de las 14 primeras iteraciones. Usamos estos contadores para diversificar la búsqueda, conduciéndola a nuevas regiones y rompiendo el ciclado. Nuestro uso de información de frecuencia penalizará movimientos de no mejora mediante la asignación de una penalización mayor a intercambios de pares de tareas con mayores contadores de frecuencia. (Típicamente, estos contadores serían normalizados, por ejemplo mediante la división por el número total de iteraciones o su

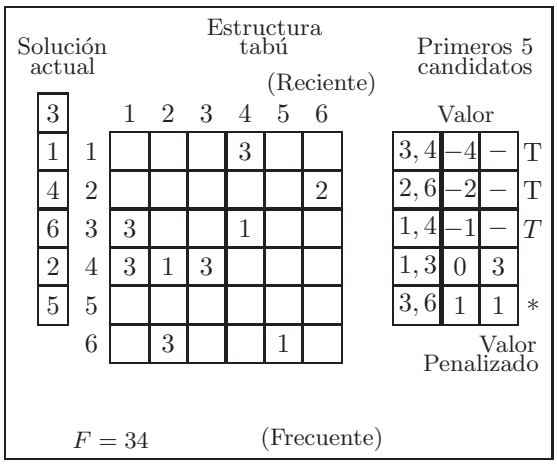


Figura 12: Iteración 15

máximo valor). Esto se ilustra en el ejemplo presente simplemente sumando el valor de frecuencia al valor del movimiento asociado.

La lista de candidatos superiores para la iteración 15 muestra que el movimiento de máxima mejora es el intercambio $(3, 4)$, pero dado que este par tiene un período tabú residual, es clasificado tabú. Lo mismo sucede con los movimientos $(2, 6)$ y $(1, 4)$. El movimiento $(1, 3)$ tiene un valor de 0, y pudiera ser en otro caso el siguiente preferido, excepto si sus tareas asociadas han sido intercambiadas frecuentemente durante la historia de la búsqueda (de hecho, más frecuentemente que cualquier otro par de tareas). Por lo tanto, el movimiento es penalizado fuertemente y pierde su atractivo. El intercambio de las tareas 3 y 6 es, por tanto, seleccionado como el mejor movimiento en la iteración actual.

La estrategia de imponer penalizaciones bajo condiciones particulares se usa para preservar la agresividad de la búsqueda. Las funciones de penalización en general se diseñan para justificar no sólo frecuencias sino también valores de movimientos y ciertas medidas de influencia.

Además, las frecuencias definidas sobre diferentes subconjuntos de soluciones anteriores, particularmente subconjuntos de soluciones élite formados por óptimos locales de alta calidad, dan lugar a estrategias complementarias llamadas estrategias de intensificación. Las estrategias de intensificación y diversificación interactúan para proporcionar puntos de apoyo fundamentales de memoria de largo plazo en búsqueda tabú. El modo en el que tales elementos son capaces de crear métodos realzados de la búsqueda, extendiendo el enfoque simplificado del ejemplo precedente, se elabora en las siguientes secciones.

3 Fundamentos de la Búsqueda Tabú: Memoria a Corto Plazo

Antes de comenzar a detallar los fundamentos de la búsqueda tabú, es necesario disponer de algunas definiciones y convenciones básicas. Expresemos un problema de optimización matemática de la siguiente forma:

$$\min c(x)$$
$$\text{Sujeto a } x \in X$$

La función objetivo $c(x)$ puede ser lineal o no lineal, y la condición $x \in X$ resume las restricciones impuestas sobre el vector x. Estas restricciones pueden incluir desigualdades lineales o no lineales, y pueden forzar a algunas o a todas las componentes de x a tomar valores discretos.

En muchas aplicaciones de optimización combinatoria, el problema de interés no es explícitamente formulado como lo hemos mostrado. En estos casos, esta formulación puede ser concebida como un código para otra formulación. El requerimiento $x \in X$, por ejemplo, puede especificar condiciones lógicas o interconexiones que sería difícil formular matemáticamente, y que es mejor dejarlas como estipulaciones verbales (por ejemplo, en forma de reglas). En ocasiones, en estas instancias, las variables son simplemente códigos para condiciones o asignaciones que reciben un valor de uno para codificar la asignación de un elemento u a una posición v, y que recibe un valor de cero para indicar que no se produce tal asignación.

3.1 Búsqueda por entorno

La búsqueda tabú puede ser caracterizada mediante referencia a la búsqueda por entornos, aunque es importante destacar que la búsqueda en el entorno tiene un significado más amplio en búsqueda tabú que en algunas otras estrategias de la literatura de las metaheurísticas. Una representación de búsqueda por entorno identifica, para cada solución $x \in X$, un conjunto asociado de vecinos, $N(x) \subset X$, llamado entorno de x. En búsqueda tabú, los entornos normalmente se asumen simétricos, es decir, x' es un vecino de x si y sólo si x es un vecino de x'. Los pasos en la búsqueda por entorno se muestran en la Figura 13.

3.2 Memoria y Clasificaciones Tabú

La idea de explotar ciertas formas de memoria adaptativa para controlar el proceso de la búsqueda es el tema central subyacente en la búsqueda tabú. Una diferencia importante que surge en búsqueda tabú es la distinción entre memoria a corto plazo y memoria a largo plazo. Cada uno de estos tipos de memoria va acompañado de sus propias estrategias especiales. Sin embargo, el efecto de ambos

Método de Búsqueda en el Entorno:

Paso 1 (Inicialización).

(A) Seleccionar una solución de arranque $xActual \in X$.

(B) Almacenar la mejor solución actual conocida haciendo $xMejor = xActual$ y definiendo $MejorCoste = c(xMejor)$.

Paso 2 (Elección y finalización).

Elegir una solución $xSiguiente \in N(xActual)$. Si los criterios de elección empleados no pueden ser satisfechos por ningún miembro de $N(xActual)$, o si se aplican otros criterios de parada, entonces el método para.

Paso 3 (Actualización).

Rehacer $xActual = xSiguiente$, y si $c(xActual) < MejorCoste$, ejecutar el paso 1(B). Volver al paso 2.

Figura 13: Método de Búsqueda en el Entorno.

tipos de memoria puede verse como la modificación de la estructura de entorno de la solución actual. El efecto de tal memoria puede ser previsto estipulando que la búsqueda tabú mantiene una historia selectiva H de los estados encontrados durante la búsqueda, y reemplaza $N(xActual)$ por un entorno modificado que puede ser denotado como $N(H, xActual)$. La historia determina, por tanto, qué soluciones pueden ser alcanzadas por un movimiento desde la solución actual, seleccionando $xSiguiente$ de $N(H, xActual)$.

En las estrategias TS basadas en consideraciones de período corto o memoria a corto plazo, $N(H, xActual)$ es generalmente un subconjunto de $N(xActual)$, y la clasificación tabú sirve para identificar elementos de $N(xActual)$ excluidos de $N(H, xActual)$. En las estrategias de período intermedio y largo, $N(H, xActual)$ puede contener soluciones que no estén en $N(xActual)$, generalmente soluciones élite seleccionadas (óptimos locales de alta calidad), encontradas durante el proceso de búsqueda. Estas soluciones élite se identifican típicamente como elementos de un grupo local en estrategias de intensificación de período intermedio, y como elementos de diferentes grupos en estrategias de diversificación de período largo o largo plazo. Además, las componentes de las soluciones élite, en contraste con las soluciones en sí mismas, se incluyen entre los elementos que pueden ser conservados e integrados para proporcionar entradas al proceso de búsqueda.

Un proceso de búsqueda local basado únicamente en estrategias a corto plazo puede permitir que una solución sea visitada más de una vez, pero es probable que el entorno reducido sea diferente en cada una de las exploraciones. Cuando

la memoria a corto plazo va acompañada de memoria a largo plazo, se reduce en gran medida la probabilidad de tomar decisiones que visiten repetidamente sólo un subconjunto limitado del espacio de soluciones.

La búsqueda tabú también usa historia para generar una función de evaluación modificada de las soluciones accesibles. Formalmente, podemos expresarlo diciendo que TS reemplaza la función objetivo $c(x)$ por una función $c(H, x)$, que tiene el propósito de evaluar la calidad relativa de las soluciones accesibles actualmente, en función de la historia del proceso. Esta función modificada es relevante porque TS usa criterios de decisión agresivos que buscan un mejor $xSiguiente$, es decir, que proporcionan un mejor valor de $c(H, xSiguiente)$, sobre un conjunto candidato de $N(H, xAhora)$.

Para problemas grandes, donde $N(H, xActual)$ puede tener muchos elementos, o para problemas donde estos elementos pueden ser costosos de examinar, la orientación de elección agresiva de TS hace altamente importante aislar un subconjunto candidato del entorno, y examinar este subconjunto en vez del entorno completo. Esto puede realizarse en etapas, permitiendo que el subconjunto candidato se extienda si no se encuentran alternativas que satisfagan los niveles de aspiración. Debido a la importancia del papel del subconjunto candidato, nos referimos a este subconjunto explícitamente por la notación $Candidato\_N(xActual)$. Entonces, el procedimiento de búsqueda tabú puede ser expresado como se muestra en la Figura 14.

Método de Búsqueda Tabú:

Paso 1 (Inicialización).
Comenzar con la misma inicialización usada para la Búsqueda por Entorno, y empezar con el expediente de la historia H vacío.

Paso 2 (Elección y finalización).
Determinar $Candidato\_N(xActual)$ como un subconjunto de $N(H, xActual)$. Seleccionar $xSiguiente$ de $Candidato\_N(xActual)$ para minimizar $c(H, x)$ sobre este conjunto ($xSiguiente$ es llamado elemento de evaluación mayor de $Candidato\_N(xActual)$). Terminar mediante un criterio de parada seleccionado.

Paso 3 (Actualización).
Ejecutar la actualización por el Método de Búsqueda en el Entorno, y actualizar el expediente de la historia H.

Figura 14: Método de Búsqueda Tabú.

La esencia del método de búsqueda tabú depende de cómo se defina y uti-

lice la historia almacenada H, y de cómo se determinen el entorno candidato $Candidato\_N(xActual)$ y la función evaluación $c(H, x)$. En los casos más simples, que suponen gran parte de las implementaciones que aparecen en la literatura, podemos considerar que $Candidato\_N(xActual)$ constituye todo $N(H, xActual)$, y que $c(H, x) = c(x)$, ignorando enfoques de investigación del entorno y la memoria a largo plazo que introduce soluciones élite en la determinación de los movimientos. Sin embargo, las estrategias de listas de candidatos que reducen el espacio de movimientos considerados son enormemente importantes para una implementación efectiva [8].

Las funciones de memoria a corto plazo constituyen uno de los elementos más importantes de la metodología de búsqueda tabú. Estas funciones aportan a la búsqueda la oportunidad de continuar más allá de la optimalidad local permitiendo la ejecución de movimientos de no mejora ligados a la modificación de la estructura de entorno de las siguientes soluciones. Sin embargo, en vez de almacenar soluciones completas, como en el enfoque de memoria explícita, estas estructuras de memoria generalmente están basadas en el almacenamiento de atributos (*memoria atributiva*). Además, la memoria a corto plazo suele estar basada en la historia reciente de la trayectoria de búsqueda.

Memoria Atributiva

Un atributo de un movimiento de $xActual$ a $xSiguiente$, o de un movimiento ensayo de $xActual$ a una solución tentativa $xEnsayo$, puede abarcar cualquier aspecto que cambie como resultado del movimiento. Algunos tipos naturales de atributos aparecen en la Figura 15.

Atributos de Movimiento Ilustrativos

para un Movimiento $xActual$ **a** $xEnsayo$**:**

(A1) Cambio de una variable seleccionada x_j de 0 a 1.

(A2) Cambio de una variable seleccionada x_k de 1 a 0.

(A3) El cambio combinado de (A1) y (A2) tomados juntos.

(A4) Cambio de una función $g(xActual)$ a $g(xEnsayo)$ (donde g puede representar una función que ocurre naturalmente en la formulación del problema o una función que es creada estratégicamente).

Figura 15: Atributos de Movimiento Ilustrativos.

Un movimiento simple evidentemente puede dar lugar a atributos múltiples. Por ejemplo, un movimiento que cambia los valores de dos variables simultáneamente

puede dar lugar a cada uno de los tres atributos (A1), (A2), y (A3), además de otros atributos de la forma indicada.

Cuando nos referimos a asignar valores alternativos a una variable seleccionada x_j de x, y particularmente a asignar valores 0 y 1 a una variable binaria, entenderemos que esto puede referirse a una variedad de operaciones tales como añadir o eliminar aristas de un grafo, asignar o eliminar un servicio de una localización particular, cambiar la posición de procesamiento de un trabajo sobre una máquina, y así sucesivamente.

Los atributos de movimientos almacenados son a menudo usados en búsqueda tabú para imponer restricciones, que evitan que sean elegidos ciertos movimientos que invertirían los cambios representados por estos atributos. Más precisamente, cuando se ejecuta un movimiento de $xActual$ a $xSiguiente$ que contiene un atributo e, se mantiene un registro para el atributo inverso que denotamos por $\bar{e}$, para prevenir que ocurra un movimiento que contenga algún subconjunto de tales atributos inversos. En la Figura 16 se muestran algunos tipos de restricciones tabú empleadas frecuentemente.

Restricciones Tabú Ilustrativas.

Un movimiento es tabú si:

(R1) x_j cambia de 1 a 0 (donde x_j cambió previamente de 0 a 1).

(R2) x_k cambia de 0 a 1 (donde x_k cambió previamente de 1 a 0).

(R3) Ocurre al menos una de las restricciones (R1) y (R2). (Esta condición es más restrictiva que (R1) o (R2) separadamente, es decir, hace más movimientos tabú).

(R4) Ocurren (R1) y (R2). (Esta condición es menos restrictiva que (R1) o (R2) por separado, es decir, hace menos movimientos tabú).

Figura 16: Restricciones Tabú Ilustrativas.

3.3 Memoria basada en lo Reciente

La memoria a corto plazo más utilizada generalmente en la literatura almacena los atributos de las soluciones que han cambiado en el pasado reciente. Este tipo de memoria a corto plazo se denomina *memoria basada en lo reciente*. La forma más habitual de explotar este tipo de memoria es etiquetando los atributos seleccionados de soluciones visitadas recientemente como tabú-activos. Se considera que un atributo es *tabú-activo* cuando su atributo inverso asociado ha ocurrido

dentro de un intervalo estipulado de lo reciente. Un atributo que no es tabú-activo se llama tabú-inactivo. De este forma, aquellas soluciones que contengan atributos tabú-activos, o combinaciones particulares de los mismos, se convierten en soluciones tabú o prohibidas. Tal como hemos mencionado anteriormente, esto impide que se visiten soluciones ya evaluadas en el pasado reciente.

La condición de ser tabú-activo o tabú-inactivo se llama el *estado tabú* de un atributo. En algunas ocasiones un atributo se llama tabú o no tabú para indicar que es tabú-activo o tabú-inactivo. Es importante tener en cuenta que un movimiento puede contener atributos tabú-activos, pero no ser tabú en sí mismo si estos atributos no son del número o clase correctas para activar una restricción tabú.

Aunque las restricciones tabú más comunes, cuyos atributos son los inversos de aquellos que definen las restricciones, tienen generalmente el objetivo de prevenir el ciclado, es necesario precisar que el objetivo final de la búsqueda tabú no es evitar ciclos. Es importante tener en cuenta que en algunas instancias, un buen camino de búsqueda resultará en volver a visitar una solución encontrada anteriormente. El objetivo más general es continuar estimulando el descubrimiento de nuevas soluciones de alta calidad.

3.4 Período Tabú

El uso de la memoria basada en lo reciente que aparece en la literatura con mayor frecuencia se gestiona mediante la creación de una o varias listas tabú. Estas listas almacenan los atributos tabú-activos e identifican, explícita o implícitamente, estados tabú actuales. El período tabú puede ser diferente para diferentes tipos o combinaciones de atributos, y con un mayor nivel de desarrollo, pueden variar también sobre diferentes estados del proceso de búsqueda. Estas variaciones del perído tabú de los atributos hace posible crear diferentes formas de balance entre las estrategias de memoria a corto y a largo plazo.

Por lo tanto, para determinar cuándo son aplicables determinadas restricciones tabú, obtenidas a partir de los estados tabú de deteminados atributos, es necesario disponer de funciones de memoria que permitan almacenarlos eficaz y eficientemente. Dos ejemplos de funciones de memoria basadas en lo reciente, usadas frecuentemente en la literatura, se especifican mediante los vectores $ComienzoTabu(e)$ y $FinTabu(e)$, donde e varía sobre atributos relevantes a una aplicación particular. Estos vectores identifican, respectivamente, las iteraciones de comienzo y finalización del período tabú para el atributo e, acotando así el período durante el cual e es tabú-activo.

La regla para identificar valores apropiados para $ComienzoTabu(e)$ y $FinTabu(e)$ resulta de mantener los atributos en cada iteración que son componentes del movimiento actual. En particular, en la iteración i, si e es un atributo del movimiento actual, se define un estado tabú para evitar inversiones. Enton-

ces $ComienzoTabu(e) = i + 1$, indicando que el atributo inverso $\bar{e}$ comienza su estado tabú-activo al comienzo de la siguiente iteración. El atributo $\bar{e}$ mantendrá este estado a lo largo de su período tabú, que denotamos por t. Esto produce $FinTabu(e) = i + t$, tal que el período para $\bar{e}$ se extiende sobre las t iteraciones de $i + 1$ a $i + t$.

Como resultado, es fácil comprobar si un atributo arbitrario es activo, simplemente controlando si $FinTabu(e) \geq IteracionActual$. Inicializando $FinTabu(e) = 0$ para todos los atributos nos aseguramos de que $FinTabu(e) < IteracionActual$, y por lo tanto que el atributo e es tabú-inactivo, hasta que se realice la actualización especificada previamente. Esto sugiere que necesitamos mantener sólo un único vector $FinTabu(e)$ para proporcionar información sobre el estado tabú. Sin embargo, veremos que surgen situaciones en las que es valioso mantener $ComienzoTabu(e)$, e inferir $FinTabu(e)$ añadiendo un valor apropiado de t (computado actualmente, o preferiblemente extraído de una secuencia prealmacenada), o mantener $FinTabu(e)$ como un vector separado.

Independientemente de la estructura de datos usada, la cuestión clave para crear el estado tabú usando memoria basada en lo reciente es determinar un "buen valor" de t o período tabú. Se ha demostrado empíricamente que un buen valor de período tabú depende del tamaño del ejemplo de problema que se aborda. Sin embargo, no se ha diseñado ninguna regla estándar que determine un período tabú efectivo para todas las clases de problemas, en parte porque un período tabú apropiado depende de la regla de activación tabú usada. Para una determinada clase de problemas es relativamente sencillo determinar períodos tabú y reglas de activación adecuadas mediante experimentación. Es posible reconocer que un período tabú es muy pequeño para una clase de problemas cuando se detectan repetitivos valores de la función objetivo, lo cual sugiere la aparición de ciclado en el proceso de búsqueda. De la misma forma, se detecta que un período tabú es muy grande cuando se produce un deterioro en la calidad de las soluciones encontradas. Es posible, por tanto, establecer un rango de períodos intermedios para obtener un buen comportamiento de la búsqueda. Una vez obtenido este rango de períodos tabú, un modo de proceder es seleccionar diferentes valores del rango en iteraciones diferentes.

Los elementos de la memoria a corto plazo mencionados hasta el momento, combinados con consideraciones de memoria a largo plazo, que se discutirán con más detalle en la siguiente sección, hacen de la búsqueda tabú un método con gran poder. Sin embargo, tal como podemos comprobar a partir de muchas de las aplicaciones que aparecen en la literatura, el enfoque inicial de memoria a corto plazo por sí mismo es capaz de generar soluciones de alta calidad.

Las reglas para determinar el período tabú, t, se clasifican en estáticas y dinámicas (Figura 17). Las reglas estáticas eligen un valor para t que se mantiene fijo a lo largo de la búsqueda. Las reglas dinámicas permiten que el valor de t varíe. La variación del período tabú durante el proceso de búsqueda proporciona

un método efectivo para inducir un balance entre examinar una región en detalle y mover la búsqueda hacia regiones del espacio diferentes.

Reglas Ilustrativas para Crear Período Tabú (Basado en lo Reciente)

Reglas Estáticas Elegir t como una constante tal que $t = 7$ o $t = \sqrt{n}$, donde n es una medida de la dimensión del problema.

Reglas Dinámicas

Dinámico Simple: Elegir t para variar (aleatoriamente o mediante un patrón sistemático) entre cotas t_{min} y t_{max}, tal que $t_{min} = 5$ y $t_{max} = 7$ o $t_{min} = .9\sqrt{n}$ y $t_{max} = 1.1\sqrt{n}$.

Dinámico Atributo Dependiente: Elegir t como en la regla dinámica simple, pero determinar t_{min} y t_{max} para ser mayores para aquellos atributos que son más atractivos; por ejemplo, basados en consideraciones de calidad o de influencia.

Figura 17: Reglas Ilustrativas para Crear Período Tabú.

Los valores indicados, tales como 7 y $\sqrt{n}$, son sólo para propósitos ilustrativos, y representan parámetros cuyos valores preferidos deberían ser establecidos mediante experimentación para una clase particular de problemas, tal como hemos indicado anteriormente. En ocasiones, es apropiado permitir que diferentes tipos de atributos definiendo una restricción tabú tengan diferentes valores de período tabú. Por ejemplo, algunos atributos pueden contribuir más fuertemente a una restricción tabú que otros, y debería asignárseles un período tabú más pequeño para impedir hacer la restricción demasiado severa.

Para ilustrarlo, consideremos el problema de identificar un subconjunto óptimo de m ítems de un conjunto mucho mayor de n ítems. Supongamos que cada movimiento consiste en intercambiar uno o un número pequeño de ítems en el subconjunto con un número igual fuera del subconjunto, para crear un nuevo subconjunto de m ítems. Además de esto, supongamos también que se usa una restricción tabú para prohibir un movimiento si contiene un ítem añadido recientemente o recientemente eliminado, donde el período tabú proporciona el significado de recientemente.

Si el período para ítems añadidos o eliminados es el mismo, la restricción anterior puede ser muy ladeada. En particular, cuando otros factores son iguales, evitar eliminar ítems del subconjunto es mucho más restrictivo que evitar que sean añadidos ítems que no están en el mismo, dado que hay menos contenido que en el subconjunto exterior. Además, evitar que elementos añadidos al subconjunto sean eliminados por un tiempo relativamente largo puede inhibir significativamente

las opciones disponibles y, por lo tanto, el período para estos elementos debería ser pequeño en comparación al período para evitar que sean añadidos elementos eliminados del subconjunto, usando reglas estáticas o dinámicas.

3.5 Criterios de Aspiración

Otro de los elementos fundamentales que permite al método de búsqueda tabú alcanzar sus mejores niveles de ejecución es la introducción de los criterios de aspiración durante el proceso de búsqueda. Los criterios de aspiración se introducen para determinar cuándo se pueden reemplazar las restricciones tabú, eliminando así la clasificación tabú aplicada a un movimiento. Aunque gran parte de las aplicaciones que encontramos en la literatura emplean únicamente un tipo simple de criterio de aspiración, que consiste en eliminar una clasificación tabú de un movimiento de ensayo cuando el movimiento conduce a una solución mejor que la mejor obtenida hasta ahora, hay otros criterios de aspiración efectivos para mejorar la búsqueda.

Una base para uno de estos criterios surge al introducir el concepto de *influencia*, que mide el grado de cambio inducido en la estructura de la solución o en la factibilidad. Esta noción puede ser ilustrada para el problema de distribuir objetos desigualmente pesados entre cajas, donde el objetivo es dar a cada caja aproximadamente el mismo peso. Un movimiento de alta influencia, que cambia significativamente la estructura de la solución actual, se ejemplifica mediante un movimiento que transfiera un objeto muy pesado de una caja a otra. Tal movimiento puede no mejorar la solución actual, siendo menos probable conducir a una mejora cuando la solución actual sea relativamente buena. Se realizarán movimientos de baja influencia mientras existan posibilidades de mejora significantes. En el momento en el que se carezca de movimientos de mejora, los criterios de aspiración cambian para dar un mayor peso a los movimientos influyentes. Además, una vez que se ha realizado un movimiento influyente, cabe pensar que nos hemos desplazado a una región diferente del espacio de búsqueda y, por tanto, deberían eliminarse las restricciones tabú establecidas previamente para movimientos menos influyentes. Estas consideraciones de influencia de movimiento interactúan con consideraciones de región y dirección de búsqueda.

Distinguimos entre *aspiraciones de movimiento* y *aspiraciones de atributo*. Cuando se satisface una aspiración de movimiento, se revoca la clasificación tabú del movimiento. De la misma forma, cuando se sastisface una aspiración de atributo, se revoca el estado tabú-activo del atributo. En el último caso el movimiento principal puede no cambiar su clasificación tabú, dependiendo de si la restricción tabú se activa a partir de más de un atributo.

Los siguientes criterios determinan la admisibilidad de una solución ensayo, $xEnsayo$, como un candidato a ser considerado, donde $xEnsayo$ es generado por un movimiento que ordinariamente sería clasificado tabú.

Criterios de Aspiración Ilustrativos

Aspiración por Defecto: Si todos los movimientos disponibles están clasificados tabú, y no se han hecho admisibles mediante algunos otros cirterios de aspiración, entonces se selecciona el movimiento "menos tabú". (Por ejemplo, seleccionamos un movimiento que pierda su clasificación tabú por el menor incremento en el valor de *IteraciónActual*, o por una aproximación a esta condición).

Aspiración por objetivo:

Forma Global: Se satisface una aspiración de movimiento, permitiendo que $xEnsayo$ sea un candidato para la selección, si $c(xEnsayo) < MejorCoste$.

Forma Regional: Subdividimos el espacio de búsqueda en regiones $R \subseteq X$, identificadas mediante cotas sobre los valores de funciones $g(x)$ (o por intervalos de tiempo de búsqueda). Denotemos por $MejorCoste(R)$ el mínimo $c(x)$ encontrado en R. Entonces para $xEnsayo \in R$, se satisface una aspiración de movimiento (para moverse hacia $xEnsayo$) si $c(xEnsayo) < MejorCoste(R)$.

Aspiración por Dirección de Búsqueda: Sea $direccion(e) = mejora$ si el movimiento más reciente conteniendo a $\bar{e}$ fue un movimiento de mejora, y $direccion(e) = nomejora$, en otro caso. ($direccion(e)$ y $FinTabu(e)$ se fijan a sus valores actuales en la misma iteración). Se satisface una aspiración de atributo para e (haciendo a e tabú-inactivo) si $direccion(e) = mejora$ y el movimiento ensayo actual es un movimiento de mejora, es decir, si $c(xEnsayo) < c(xActual)$.

Aspiración por Influencia: Sea $influencia(e) = 0$ ó 1 según si el movimiento que establece el valor de $ComienzoTabu(e)$ es un movimiento de baja influencia o un movimiento de alta influencia. ($influencia(e)$ se fija a la vez que $ComienzoTabu(e)$). Además, sea $Ultima(L)$, para $L = 0$ ó 1, igual a la iteración más reciente en la que fue realizado un movimiento de nivel de influencia L. Entonces una aspiración de atributo para e se satisface si $influencia(e) = 0$ y $ComienzoTabu(e) < Ultima(1)$. Para múltiples niveles de influencia $L = 0, 1, 2, ...$, la aspiración para e se satisface si hay un $L > influencia(e)$ tal que $ComienzoTabu(e) < Ultima(L)$.

Las aspiraciones por Dirección de la Búsqueda y por Influencia proporcionan aspiraciones de atributos en vez de aspiraciones de movimientos. En la mayoría de los casos, las aspiraciones de atributos y movimientos son equivalentes. Sin embargo, se emplean diversos medios para probar estas dos clases de aspiraciones.

Refinamientos de los Criterios de Aspiración

Algunas mejoras de los criterios ilustrados anteriormente proporcionan una oportunidad para realzar la potencia de la búsqueda tabú para aplicaciones que son más complejas, o que ofrecen una recompensa por soluciones de muy alta calidad. En lo que sigue identificamos algunas de las posibilidades para alcanzar esto.

La creación de un estado tabú que varíe por grados, más que simplemente señalar un atributo para ser tabú-activo o tabú-inactivo, conduce a un refinamiento adicional de Aspiración por Dirección de Búsqueda y Aspiración por Influencia. El estado tabú graduado está implícito en las variantes probabilísticas de la búsqueda tabú, donde el estado se expresa como una función de cómo un atributo se ha convertido recientemente o frecuentemente en tabú-activo y tabú-inactivo. Sin embargo, para emplear esta idea de realzar los criterios de aspiración precedentes, creamos un único estado tabú intermedio que cae entre los dos estados de tabú-activo y tabú-inactivo. En particular, cuando se satisface una aspiración para un atributo que en otro caso es tabú-activo, lo llamamos atributo tabú pendiente.

Un movimiento que sería clasificado tabú si sus atributos tabú pendientes fueran tratados como tabú-activos, pero que no sería clasificado tabú en otro caso, es llamado movimiento tabú pendiente. Un movimiento tabú pendiente puede ser tratado en uno de dos modos. En el enfoque menos restrictivo, tal movimiento no se previene de ser seleccionado, pero su estado cambia de tal manera que sólo es candidato para selección si no existen movimientos de mejora excepto aquellos que son tabú. En el enfoque más moderado, un movimiento tabú pendiente debe ser, además, un movimiento de mejora para ser calificado para selección.

Aspiración por Admisibilidad Fuerte. Las nociones precedentes conducen a un tipo adicional de aspiración. Definimos un movimiento como fuertemente admisible si:

(1) es admisible para ser seleccionado y no confía en criterios de aspiración para calificar para admisibilidad, o

(2) califica para admisibilidad basado en la Aspiración Global por Objetivo, satisfaciendo
$c(xEnsayo) < MejorCoste$.

La desigualdad $UltimaNomejora < UltimaFuertementeAdmisible$ de la condición de aspiración precedente implica, por un lado, que se ha realizado un movimiento de mejora fuertemente admisible desde el último movimiento de no mejora y, por otro lado, que actualmente la búsqueda está generando una secuencia de mejora.

Este tipo de aspiración asegura que el método siempre procederá a un óptimo local siempre que se cree una secuencia de mejora que contenga al menos un

Aspiración por Admisibilidad Fuerte: Sea $UltimaNomejora$ igual a la iteración más reciente en la que fue realizado un movimiento de no mejora, y sea $UltimaFuertementeAdmisible$ igual a la iteración más reciente en la que fue realizado un movimiento fuertemente admisible. Entonces, si $UltimaNomejora < UltimaFuertementeAdmisible$, re-clasificamos cada movimiento tabú de no mejora como un movimiento tabú pendiente (permitiendo por tanto que sea un candidato para selección si no existe otro movimiento de mejora).

movimiento fuertemente admisible. De hecho, la condición (2) que define un movimiento fuertemente admisible puede ser eliminada sin alterar este efecto, dado que una vez que se usa el criterio $c(xEnsayo) < MejorCoste$ para justificar una selección de movimiento, entonces continuará siendo satisfecho por todos los movimientos de mejora en iteraciones subsiguientes hasta que se alcance un óptimo local.

Consideraciones Especiales para la Aspiración por Influencia

El criterio de Aspiración por Influencia puede ser modificado para crear un impacto considerable sobre su efectividad para ciertos tipos de aplicaciones. La afirmación de esta aspiración deriva de la observación de que un movimiento característicamente es influyente en virtud de contener uno o más atributos influyentes. Bajo tales condiciones, es apropiado considerar niveles de influencia definidos sobre los atributos, expresado por $influencia(e)$. En otros casos, sin embargo, un movimiento puede derivar su influencia de la combinación única de los atributos involucrados, y entonces la Aspiración por Influencia preferiblemente transforma una aspiración de movimiento en vez de una aspiración de atributo.

Más significativamente, en muchas aplicaciones, la influencia depende de una forma de conectividad, haciendo a sus efectos ser expresados principalmente sobre un rango particular. Llamaremos a este rango *esfera de influencia* del movimiento o atributo asociado. Por ejemplo, en el problema de distribución de objetos entre cajas, un movimiento que intercambia objetos entre dos cajas tiene una esfera de influencia relativamente estrecha, afectando sólo a aquellos movimientos futuros que transfieran un objeto dentro o fuera de una de estas dos cajas. Por consiguiente, bajo tales circunstancias, la Aspiración por Influencia debería estar limitada a modificar el estado tabú de atributos o la clasificación tabú de los movimientos que caen dentro de una esfera de influencia asociada. En el ejemplo de intercambiar objetos entre cajas, los atributos hechos tabú-inactivo deberían ser restringidos a $DesdeAtributos$, asociados con mover un objeto fuera de una de las dos cajas y $HaciaAtributos$, asociados con mover un objeto dentro de una de estas cajas. El cambio del estado tabú continúa dependiendo de las condi-

ciones conocidas previamente. La influencia del atributo (o movimiento que lo contenga) debe ser menor que la de un movimiento anterior, y la iteración para el atributo debe preceder a la iteración sobre la cual ocurrió el movimiento influyente anterior. Estas condiciones pueden ser registradas colocando un indicador para $ComienzoTabu(e)$ cuando se ejecuta el movimiento influyente, sin tener que comprobar otra vez para ver si e es afectado por tal movimiento. Cuando a $ComienzoTabu(e)$ se le reasigna un nuevo valor, el indicador es eliminado.

Como sugieren las observaciones precedentes, son extremadamente importantes las medidas de la influencia del movimiento y las caracterizaciones asociadas de esferas de influencia. Además, debería notarse que la influencia puede ser expresada como una función de los componentes de la memoria de la búsqueda tabú, como cuando un movimiento que contiene atributos que no han sido ni frecuentemente ni recientemente tabú-activos puede ser clasificado como más altamente influyente (porque ejecutar el movimiento cambiará el estado tabú de estos atributos más radicalmente). Esto fomenta una definición dinámica de la influencia, la cual varía según el estado actual de la búsqueda.

4 Fundamentos de la Búsqueda Tabú: Memoria a largo plazo

En algunas aplicaciones, los componentes de la memoria a corto plazo son suficientes para producir soluciones de alta calidad. Sin embargo, tal como hemos mencionado anteriormente, la inclusión de la memoria a largo plazo, así como de las estrategias asociadas a la misma hacen de la búsqueda tabú una estrategia más fuerte. En las estrategias de memoria a largo plazo, los entornos modificados de las soluciones actuales pueden contener soluciones que no estén en el entorno original. Generalmente, se incluyen soluciones élite encontradas durante el proceso de búsqueda.

4.1 Memoria Basada en Frecuencia

La memoria basada en frecuencia proporciona un tipo de información que complementa la información proporcionada por la memoria basada en lo reciente, ampliando la base para seleccionar movimientos preferidos. Al igual que sucede en la memoria basada en lo reciente, la frecuencia a menudo está ponderada o descompuesta en subclases teniendo en cuenta las dimensiones de calidad de la solución e influencia del movimiento.

Concebimos medidas de frecuencia como proporciones, cuyos numeradores representan contadores del número de ocurrencias de un evento particular (por ejemplo, el número de veces que un atributo particular pertenece a una solución o movimiento) y cuyos denominadores generalmente representan uno de cuatro

tipos de valores: (1) el número total de ocurrencias de todos los eventos representados por los numeradores (tal como el número de iteraciones asociadas), (2) la suma de los numeradores, (3) el máximo valor del numerador, y (4) la media del valor del numerador. Los denominadores (3) y (4) dan lugar a lo que se puede llamar frecuencias relativas. En los casos en los que los numeradores representan cuentas ponderadas, algunas de las cuales pueden ser negativas, los denominadores (3) y (4) se expresan como valores absolutos y el denominador (2) se expresa como una suma de valores absolutos.

En el ejemplo de intercambiar objetos entre cajas, tal como indicamos anteriormente, los atributos *DesdeAtributos* están asociados con mover un objeto fuera de una de las dos cajas y los atributos $HaciaAtributos$ están asociados con mover un objeto dentro de una de estas cajas.

Denotemos por $x(1), x(2), \ldots, x(IteracionActual)$ la secuencia de soluciones generadas en el momento presente del proceso de búsqueda, y denotemos por S una subsecuencia de esta secuencia de soluciones. Tomamos la libertad de tratar S como un conjunto además de como una secuencia ordenada. Los elementos de S no son necesariamente elementos consecutivos de la secuencia de solución completa.

A modo de notación, denotemos por $S(x_j = p)$ el conjunto de soluciones en S para las cuales $x_j = p$, y denotemos por $\#S(x_j = p)$ la cardinalidad de este conjunto (el número de veces que x_j recibe el valor p sobre $x \in S$). Análogamente, denotemos por $S(x_j = p$ a $x_j = q)$ el conjunto de soluciones en S que resultan por un movimiento que cambia $x_j = p$ a $x_j = q$. Finalmente, denotemos por S(de $x_j = p$) y S(a $x_j = q$) los conjuntos de soluciones en S que contienen respectivamente $x_j = p$ como un *DesdeAtributo* o $x_j = q$ como un $HaciaAtributo$. En general, si *AtributoSolucion* representa cualquier atributo de una solución que puede tomar el papel de un *DesdeAtributo* o un $HaciaAtributo$ para un movimiento, y si $MovimientoAtributo$ representa un atributo de movimiento arbitrario denotado por $(DesdeAtributo, HaciaAtributo)$, entonces

$S(SolucionAtributo) = \{x \in S$: x contiene $AtributoSolucion\}$.

$S(MovimientoAtributo) = \{x \in S$: x resulta de un movimiento que contiene $MovimientoAtributo\}$.

$S(DesdeAtributo) = \{x \in S$: x inicia un movimiento a $DesdeAtributo\}$.

$S(HaciaAtributo) = \{x \in S$: x resulta de un movimiento que contiene a $HaciaAtributo\}$.

La cantidad $\#S(x_j = p)$ constituye una *medida de residencia*, dado que identifica el número de veces que el atributo $x_j = p$ reside en las soluciones de S. Correspondientemente, llamamos a la frecuencia que resulta de dividir tal medida por uno de los denominadores de (1) a (4) una *frecuencia de residencia*. Para el numerador $\#S(x_j = p)$, los denominadores (1) y (2) corresponden ambos a $\#S$, mientras que los denominadores (3) y (4) son dados respectivamente por $Max(\#S(x_k = q) :$ todo $k, q)$ y por $Media(\#S(x_k = q) : \forall\ k, q)$.

Las cantidades $\#S(x_j = p$ a $x_j = q)$, $\#S($de $x_j = p)$ y $\#S($a $x_j = q)$ constituyen medidas de transición, dado que identifican el número de veces que x_j cambia de y/o a valores especificados. Asímismo, las frecuencias basadas en tales medidas son llamadas *frecuencias de transición*. Los denominadores para crear tales frecuencias de las medidas precedentes incluyen $\#S$, el número total de veces que los cambios indicados ocurren sobre S para diferentes valores j, p y/o q, y cantidades Max y $Media$ asociadas.

Las frecuencias de residencia y transición en ocasiones transmiten información relacionada. Sin embargo, aunque a veces son confundidas en la literatura, en general tienen implicaciones diferentes. Una distinción significativa es que las medidas de residencia, en contraste con las medidas de transición, no se refieren a si un atributo de solución particular de un elemento $x(i)$ en la secuencia S es un $DesdeAtributo$ o un $HaciaAtributo$, o incluso si es un atributo que cambia en movimiento de $x(i)$ a $x(i+l)$ o de $x(i-l)$ a $x(i)$. Sólo es relevante que el atributo puede ser un $DesdeAtributo$ o un $HaciaAtributo$ en algún movimiento futuro. Tales medidas pueden conducir a diferentes tipos de implicaciones dependiendo de la elección de la subsecuencia de S.

Una frecuencia de residencia alta, por ejemplo, puede indicar que un atributo es altamente atractivo si S es una subsecuencia de soluciones de alta calidad, o puede indicar lo contrario si S es una subsecuencia de soluciones de baja calidad. Por otro lado, una frecuencia de residencia que es alta (baja) cuando S contiene tanto soluciones de alta como de baja calidad puede apuntar a atributo fortalecido (o excluido) que restringe al espacio de búsqueda, y que necesita ser desechado (o incorporado) para permitir diversidad.

Desde el punto de vista de la simplificación del cómputo, cuando S está formado por todas las soluciones generadas después de una iteración especificada, entonces puede mantenerse una medida de residencia actual y actualizada por referencia a valores del vector $ComienzoTabu$, sin la necesidad de incrementar un conjunto de contadores en cada iteración. Para un conjunto S cuyas soluciones no vienen de iteraciones secuenciales, sin embargo, las medidas de residencia se calculan simplemente poniendo una etiqueta sobre los elementos de S.

Las medidas de transición son generalmente bastante fáciles de mantener ejecutando actualizaciones durante el proceso de generación de soluciones (asumiendo que las condiciones que definen S, y los atributos cuyas medidas de transición son buscadas, se especifican con anterioridad). Esto resulta del hecho de que típicamente sólo se consideran relevantes unos pocos tipos de cambios de atributos para detectar cuándo una solución se reemplaza por la siguiente, y éstos pueden aislarse y registrados fácilmente. Las frecuencias del ejemplo de la sección 2.3 constituyen una instancia de frecuencias de transición que fueron mantenidas en esta manera simple. Su uso en este ejemplo, sin embargo, alentaba la diversidad aproximando el tipo de papel que las frecuencias de residencia son usualmente mejor satisfechas para ser tomadas.

Como una distinción final, una frecuencia de transición alta, en contraste con una frecuencia de residencia alta, puede indicar que un atributo asociado es un "llenador excelente", que cambia dentro y fuera de la solución para ejecutar una función de buen ajuste. Tal atributo puede ser interpretado como el opuesto de un atributo influyente, como se consideró anteriormente en la discusión de Aspiración de Influencia. En este contexto, una frecuencia de transición puede ser interpretada como una medida de volatilidad.

Ejemplos de Usos de Medidas de Frecuencia. A continuación se muestran ilustraciones de frecuencias de residencia y de transición. (Sólo se indican los numeradores, entendiendo que los denominadores son proporcionados por las condiciones (1) a (4)).

Ejemplos de Medidas de Frecuencia

(Numeradores)

(F1) $\#S(x_j = p)$
(F2) $\#S(x_j = p$ para algún $x_j)$
(F3) $\#S($a $x_j = p)$
(F4) $\#S(x_j$ cambia), es decir, $\#S(x_j \neq p$ a $x_j = p)$
(F5) $\sum_{x \in S(x_j = p)} c(x) / \#S(x_j = p)$
(F6) Reemplazar $S(x_j = p)$ en (F5) con $S(x_j \neq p$ a $x_j = p)$
(F7) Reemplazar $c(x)$ en (F6) con una medida de la influencia $S(x_j \neq p$ a $x_j = p)$

La medida (F5) puede ser interpretada como el valor medio $c(x)$ sobre S cuando $x_j = p$. Esta cantidad puede ser directamente comparada con otras medias o puede ser pasada a una medida de frecuencia usando denominadores tales como la suma o el máximo de estas medias.

Los atributos que tienen mayores medidas de frecuencia, como aquellos que tienen mayores medidas de lo reciente (es decir, que ocurrieron en soluciones o movimientos más cercanos al presente), pueden iniciar un estado tabú-activo si S está formado por soluciones consecutivas que finalizan con la solución actual. Sin embargo, la memoria basada en frecuencia típicamente encuentra su uso más productivo como parte de una estrategia de período más largo, la cual emplea incentivos además de restricciones para determinar qué movimientos son seleccionados. En tal estrategia, las restricciones se convierten en penalizaciones de evaluación, y los incentivos se convierten en mejoras de la evaluación, para alterar la base para calificar movimientos como atractivos o no atractivos.

Para ilustrarlo, a un atributo tal como $x_j = p$ con una frecuencia de residencia

alta le puede ser asignado un incentivo fuerte ("beneficio") para servir como un *DesdeAtributo*, resultando por tanto en la elección de un movimiento que produce $x_j \neq p$. Tal incentivo es particularmente relevante en el caso donde $ComienzoTabu(x_j \neq p)$ es pequeño, dado que este valor identifica la última iteración en que $x_j \neq p$ sirvió como un *DesdeAtributo*, y por tanto descubre que $x_j = p$ ha sido un atributo de cada solución desde entonces.

La memoria basada en frecuencia por tanto es usualmente aplicada introduciendo estados tabú graduados, como un fundamento para definir valores de penalización e incentivos para modificar la evaluación de los movimientos. Existe una conexión natural entre este enfoque y el enfoque de memoria basada en lo reciente que crea estados tabú como una condición todo-o-ninguno. Si el período de un atributo en memoria basada en lo reciente está concebida como un umbral condicional para aplicar una penalización muy grande, entonces las clasificaciones tabú producidas por tal memoria pueden ser interpretadas como el resultado de una evaluación que se convierte fuertemente inferior cuando las penalizaciones están activadas. Es razonable anticipar que los umbrales condicionales deberían también ser relevantes para determinar los valores de penalizaciones y los incentivos en estrategias de período largo. La mayoría de las aplicaciones en el presente, sin embargo, usan un múltiplo lineal simple de una medida de frecuencia para crear un término de penalización o de incentivo.

4.2 Estrategias de Intensificación y Diversificación Simples

Las funciones de intensificación y diversificación en la búsqueda tabú ya están implícitas en muchas de las prescripciones anteriores, pero se convierten especialmente relevantes en procesos de búsqueda de período largo. Las estrategias de intensificación crean soluciones agresivamente estimulando la incorporación de "atributos buenos". En el período corto esto consiste en incorporar atributos que han recibido las mayores evaluaciones por los enfoques y criterios descritos anteriormente, mientras que en el intermedio a largo período consiste en incorporar atributos de soluciones de subconjuntos élite seleccionados. Por otro lado, las estrategias de diversificación generan soluciones que incorporan composiciones de atributos significativamente diferentes a los encontrados previamente durante la búsqueda. Estos dos tipos de estrategias se contrapesan y refuerzan mutuamente de varias formas.

Examinamos formas simples de enfoques de intensificación y diversificación que hacen uso de memoria basada en frecuencia. Estos enfoques serán ilustrados por referencia a medidas de frecuencia de residencia, pero algunas observaciones similares se aplican al uso de medidas de transición, teniendo en cuenta características contrastantes notadas previamente.

Para una estrategia de diversificación elegimos S como un subconjunto significativo de la secuencia de solución completa; por ejemplo, la secuencia en-

tera empezando con el primer óptimo local, o la subsecuencia formada por todos los óptimos locales. (Para ciertas estrategias basadas en medidas de transición, S puede estar formado por la subsecuencia que contiene cada sucesión intacta máxima de movimientos de no-mejora que inmediatamente siguen un óptimo local, concentrándose en $S(HaciaAtributo)$ para estos movimientos).

Para una estrategia de intensificación elegimos S como un subconjunto pequeño de soluciones élite (óptimos locales de alta calidad) que comparten un gran número de atributos comunes, y en segundo lugar cuyos miembros pueden alcanzarse uno de otro mediante números de movimientos relativamente pequeños, independientes de si estas soluciones caen cerca la una de la otra en la secuencia de la solución. Por ejemplo, las colecciones de tales subconjuntos S pueden ser generadas por procedimientos de agrupamiento, seguido del uso de un enfoque de procesamiento paralelo para tratar cada S seleccionado por separado.

Para propósitos ilustrativos, supongamos que un movimiento actualmente bajo consideración incluye dos atributos de movimiento, denotados por e y f, los cuales pueden ser expresados como $e = (eDesde, eHacia)$ y $f = (fDesde, fHacia)$. Proporcionamos reglas para generar una función de penalización o incentivo, PI, basada en medidas de frecuencia de los atributos e y f, las cuales se aplican igualmente a estrategias de intensificación y diversificación. Sin embargo, la función PI crea una penalización para una estrategia (intensificación o diversificación) si y sólo si crea un incentivo para la otra. Para describir esta función, denotemos por $f(eDesde)$ y $f(eHacia)$, etc., la medida de frecuencia para los $DesdeAtributos$ y $HaciaAtributos$ indicados, y denotemos por $T1, T2, ..., T6$ umbrales positivos seleccionados, cuyos valores dependen del caso considerado.

Funciones PI Ilustrativas de Penalización e Incentivo para $HaciaAtributos$.

Elegir PI como una función monótona no decreciente de una de las siguientes cantidades, donde PI es positiva cuando la cantidad es positiva, y es 0 en otro caso. (PI proporciona una penalización en una estrategia de diversificación y un incentivo en una estrategia de intensificación).

(1) $Min\{f(eHacia), f(fHacia)\} - T_1$
(2) $Max\{f(eHacia), f(fHacia)\} - T_2$
(3) $Media\{f(eHacia), f(fHacia)\} - T_3$

Las condiciones precedentes para definir PI están relacionadas con las ilustradas previamente para identificar condiciones en las cuales los atributos se convierten en tabú-activos. Por ejemplo, especificando que (1) debe ser positivo para hacer PI positivo corresponde a introducir una penalización tabú (o un incentivo) cuando ambas medidas exceden sus umbrales comunes. Si una medida

Funciones PI Ilustrativas de Penalización e Incentivo para *DesdeAtributos*.

Elegir PI como una función monótona no decreciente de una de las siguientes cantidades, donde PI es positiva cuando la cantidad es positiva, y es 0 en otro caso. (PI proporciona un incentivo en una estrategia de diversificación y una penalización en una estrategia de intensificación).

(1) $Min\{f(eDesde), f(fDesde)\} - T_4$
(2) $Max\{f(eDesde), f(fDesde)\} - T_5$
(3) $Media\{f(eDesde), f(fDesde)\} - T_6$

es expresada como la duración desde que un atributo fue el más recientemente hecho tabú-activo, y si el umbral representa un límite común para el período tabú, entonces (1) puede expresar una restricción basada en lo reciente para determinar una clasificación tabú. La asignación de diferentes umbrales a atributos diferentes en (1) corresponde a establecer períodos tabú atributo-dependientes. Análogamente, los restantes valores de (2) a (6) pueden ser interpretados como análogos a los valores que definen medidas basadas en lo reciente para establecer una clasificación tabú, implementada en este caso a través de una penalización.

De estas observaciones se concluye que la medida de frecuencia F puede extenderse para representar medidas combinadas de lo reciente y de lo frecuente. Note que la memoria basada en lo reciente, almacenando datos de $ComienzoTabu$, puede también referirse a cambios que han ocurrido más lejos en el pasado además de aquellos que han ocurrido más recientemente. Aunque estas medidas están ya implícitamente combinadas cuando se unen las penalizaciones y los incentivos basados en medidas de frecuencia con clasificaciones tabú basadas en medidas de lo reciente, como un fundamento para seleccionar movimientos actuales, es posible que otras formas de combinación sean superiores.

4.3 Aspectos más avanzados de Intensificación y Diversificación

Los métodos de intensificación y diversificación que utilizan penalizaciones e incentivos representan sólo una clase de tales estrategias. Una colección mayor surge de la consideración directa de los objetivos de intensificación y diversificación. Examinamos diversos métodos que se han demostrado útiles en aplicaciones previas, e indicamos métodos que consideramos prometedores en aplicaciones futuras. Para empezar hacemos una distinción importante entre diversificación y aleatorización.

Diversificación frente a aleatorización. Cuando la búsqueda tabú busca una

colección de soluciones diversas, es muy diferente de cuando busca una colección de soluciones aleatorias. En general, estamos interesados no sólo en colecciones diversas sino en secuencias diversas, dado que frecuentemente el orden en el que se examinan los elementos es importante en TS. Esto ocurre, por ejemplo, cuando buscamos identificar una secuencia de nuevas soluciones de forma que cada solución sucesiva sea *maximalmente* diversa en relación a todas las soluciones previamente generadas. Esto incluye posibles referencias a un conjunto base de soluciones, tales como $x \in S$, que da prioridad al objetivo de diversificación (es decir, donde el primer objetivo es establecer diversificación con respecto a S, y después con respecto a otras soluciones generadas). El concepto de diversificación se aplica también a la generación de una secuencia diversa de números o a un conjunto diverso de puntos entre los vértices del hipercubo unidad. Sea $Z(k) = (z(1), z(2), ..., z(k))$ una secuencia de puntos del conjunto Z. Por ejemplo, Z puede ser un intervalo lineal si los puntos son escalares. Tomamos $z(1)$ como *punto semilla* de la secuencia. Entonces definimos $Z(k)$ como una *secuencia dispersa* relativa a una métrica de distancia d elegida sobre Z requiriendo que cada subsecuencia $Z(h)$ de $Z(k)$, $h \leq k$, en todo punto asociado $z = z(h+1)$ satisfaga las siguientes condiciones jerárquicas:

(A) z maximiza la distancia mínima $d(z, z(i))$ para $i \leq h$;

(B) sujeto a (A), z maximiza la distancia mínima $d(z, z(i))$ para $1 < i \leq h$, para $2 < i \leq h$, etc. (en orden de prioridad estricto);

(C) sujeto a (A) y (B), z maximiza la distancia mínima $d(z, z(i))$ para $i = h$, para $i = h - 1$,..., y finalemte para $i = 1$. (Los empates pueden resolverse arbitrariamente.)

Para tratar la diversificación relativa a un conjunto base inicial Z^* (tal como un conjunto de soluciones $x \in S$), la jerarquía precedente de condiciones se precede por una condición que estipula que z primero maximiza la mínima distancia $d(z, z^*)$ para $z^* \in Z^*$. Una variante (más débil) útil de esta condición trata simplemente puntos de Z^* como si fueran los últimos elementos de la secuencia $Z(h)$.

Algunas variaciones sobre (A), (B) y (C), incluso profundizando en la jerarquía anterior (desempates arbitrarios), son evidentemente posibles. Además, computacionalmente demandan ser satisfechas. Incluso omitiendo (B), y manteniendo sólo (A) y (C), si los elementos $z(i)$ se refieren a puntos del hipercubo unidad, entonces según nuestro conocimiento actual, la única manera de generar una secuencia diversa de más de unos pocos puntos es ejecutar una enumeración comparativa. (No obstante, una secuencia dispersa de puntos en un intervalo lineal, particularmente si $z(1)$ es un extremo o el punto medio del intervalo, se puede generar sin mucha dificultad). Con una visión más amplia, el esfuerzo que requiere la generación de secuencias dispersas puede llevarse acabo previamente

e independientemente de los esfuerzos para resolver el problema, con lo que tales secuencias están precalculadas y disponibles cuando se necesiten.

Refuerzo por restricción. Uno de los primeros tipos de estrategias de intensificación, caracterizada en términos de explotar variables fuertemente determinadas y consistentes en [4], comienza seleccionando un conjunto S como indicado para determinar una penalización y una función de incentivo, es decir, un conjunto formado por soluciones élite agrupadas a través de una medida de clasificación. En vez de (o además de) crear penalizaciones e incentivos, con el objetivo de incorporar atributos a la solución actual que tenga altas medidas de frecuencia sobre S, el método de refuerzo por restricción opera estrechando el rango de posibilidades permitidas añadiendo y quitando tales atributos. Por ejemplo si $x_j = p$ tiene una alta frecuencia sobre S sólo para un pequeño número de valores de p, entonces los movimientos se restringen permitiendo a x_j tomar sólo uno de estos atributos en la definición de un $HaciaAtributo$. Por tanto, si x_j es una variable 0-1 con una medida de frecuencia alta sobre S para uno de sus valores, entonces este valor se hará fijo una vez que exista un movimiento admisible que permita que se asigne dicho valor. Otras asignaciones pueden permitirse, por una variante de Aspiración por Defecto, si el conjunto actual de alternativas restringidas es inaceptable.

La consideración inicial sugiere que este método de restricción no ofrece nada más allá de las opciones disponibles por penalizaciones e incentivos. No obstante, el método puede conseguir más que esto por dos motivos. Primero, las restricciones explícitas pueden acelerar substancialmente la ejecución de los pasos de elección reduciendo el número de alternativas examinadas. Segundo, y más significativamente, muchos problemas se simplifican y colapsan una vez que se introducen un número de restricciones explícitas, permitiendo que las implicaciones estructurales salgan a la superficie, permitiendo que estos problemas se resuelvan más fácilmente.

El refuerzo por restricción no se limita a crear un efecto de intensificación. Dados energía y tiempo finitos para explorar alternativas, imponer restricciones a algunos atributos permite examinar más variantes de los restantes atributos que de otra manera. Por tanto, la intensificación con respecto a los elementos seleccionados puede realzar la diversificación sobre otros elementos, creando una forma de diversificación selectiva. Tal diversificación puede contrastarse con diversificación exhaustiva creada por las estructuras de memoria más rígidas de ramificación y acotación. En un ambiente donde el aspecto finito del esfuerzo de búsqueda disponible es proporcionado por el número de alternativas a ser exploradas exhaustivamente, la diversificación selectiva puede ser una contribución significativa a la búsqueda efectiva.

Reencadenamiento de camino. El reencadenamiento de camino (PR, *path relinking*) se inicia seleccionando dos soluciones x' y x'' de una colección de soluciones élite producidas durante las fases de búsqueda. Se genera un camino desde

x' a x'', produciendo una secuencia de soluciones $x' = x'(1), x'(2), ..., x'(r) = x''$ donde $x'(i+1)$ se crea a partir de $x'(i)$ en cada paso eligiendo un movimiento que deja el menor número de movimientos restantes hasta alcanzar x''. Finalmente, una vez que el camino esté completo, una o más de las soluciones $x'(z)$ se seleccionan como soluciones para iniciar una nueva fase de búsqueda

Este método proporciona un medio fundamental para perseguir el objetivo de intensificación y diversificación cuando sus pasos se implementan para explotar variantes estratégicas de reglas de elección. Un número de movimientos alternativos típicamente calificarán para producir la siguiente solución a partir de $x'(i)$ por el critero del "menor número de movimientos restantes", permitiendo consecuentemente una variedad de caminos posibles de x' a x''. Seleccionar movimientos no atractivos relativos a $c(x)$ en cada paso tenderá a producir una serie final de movimientos de fuerte mejora, mientras que seleccionar movimientos atractivos tenderá a producir movimientos de menor calidad al final. (El último movimiento, no obstante, mejorará, o dejará $c(x)$ sin cambiar, ya que x'' es un mínimo local.) Por tanto, elegir el mejor, peor o movimiento medio, usando un criterio de aspiración para anular las elecciones en los dos últimos casos si está disponible una solución suficientemente atractiva, proporciona opciones que producen efectos contrastantes en la generación de la secuencia indicada. (Existen argumentos a favor de seleccionar el mejor movimiento en cada paso, y entonces repetir el proceso intercambiando x' y x''.)

La cuestión de una aspiración apropiada más amplia es relevante para seleccionar una $x'(i)$ preferida para lanzar una nueva fase de búsqueda, y para terminar la secuencia más pronto. La elección de una o más soluciones $x'(z)$ para lanzar una nueva fase de búsqueda debe depender preferiblemente no sólo de $c(x'(i))$ sino también de los valores de $c(x)$ de aquellas soluciones x que pueden alcanzarse por un movimiento a partir de $x'(i)$. En particular, cuando $x'(i)$ se examina para moverse a $x'(i+1)$, se presentará un número de candidatos para $x = x'(i+1)$ para su consideración.

Sea $x^*(i)$ un vecino de $x'(i)$ que proporciona un mínimo valor de $c(x)$ durante un paso de evaluación, excluyendo $x^*(i) = x'(i+1)$. (Si las reglas de elección no eliminan automáticamente la posibilidad $x^*(i) = x'(h)$ para $h < i$, entonces una simple restricción tabú puede usarse para esto). Entonces el método selecciona una solución $x^*(i)$ que da el valor mínimo para $c(x^*(i))$ como un nuevo punto para lanzar la búsqueda. Si sólo se examina un conjunto limitado de vecinos de $x'(i)$ para identificar $x^*(i)$, entonces se puede seleccionar en su lugar un $x'(i)$ de coste mínimo superior, excluyendo x' y x''. Una terminación temprana puede ser elegida al encontrar un $x^*(i)$ que de $c(x^*(i)) < min\{c(x'), c(x"), c(x'(p))\}$, donde $x'(p)$ es el $x'(h)$ de mínimo coste para todo $h \leq i$. (El procedimiento continúa sin parar si $x'(i)$, en cotraste con $x^*(i)$, da un valor $c(x)$ menor que x' y x'', ya que $x'(i)$ adopta efectivamente el papel de x').

Variaciones y Túneles. Una variante del método PR empieza desde ambos ex-

tremos x' y x'' simultáneamente, produciendo dos secuencias $x' = x'(1), ..., x'(r)$ y $x" = x"(1), ..., x"(s)$. Las elecciones se diseñan para provocar que $x'(r) = x''(s)$ para los valores finales de r y s. Para progresar hacia este resultado cuando $x'(r) \neq x''(s)$, se selecciona $x'(r)$ para crear $x'(r+1)$, mediante el criterio de minimizar el número de movimientos restantes hasta alcanzar $x''(s)$, o se selecciona $x''(s)$ para crear $x''(s+1)$, mediante el criterio de minimizar el número de movimientos restantes hasta alcanzar $x'(r)$. De estas opciones se selecciona la que produzca el menor valor $c(x)$, determinando también si r o s se incrementa en el paso siguiente.

El método de re-encadenamiento de camino se puede beneficiar de un procedimiento de efecto "túnel" que permita usar una estructura de entornos diferente que la de la fase de búsqueda estándar. En particular, frecuentemente es deseable permitir periódicamente movimientos para el reenlace de camino que normalmente se excluirían por crear infactibilidad. Esta práctica es menos susceptible de llegar a perderse en una región no factible que otras formas de permitir infactibilidad periódica, ya que evidentemente la factibilidad se vuelve a recuperar al llegar a x''. El efecto túnel creado ofrece la oportunidad para alcanzar soluciones que de otra forma se pasarían por alto. En la variante que empieza desde x' y x'', algunas de las soluciones de $x'(r)$ o $x''(s)$ deben mantenerse factibles.

El re-encadenamiento de camino se puede organizar para poner más énfasis en la intensificación o diversificación optando por que x' y x'' compartan más o menos atributos. Análogamente la elección de x' y x'' de un conjunto clasificado de soluciones élite estimulará la intensificación, mientras que elegirlas de conjuntos ampliamente separados estimulará la diversificación.

Re-encadenamiento extrapolado. Una extensión del método del re-encadenamiento de camino, que llamamos *re-encadenamiento extrapolado*, va más allá del punto extremo x'' (o alternativamente x'), para obtener soluciones que se expanden a una región mayor. La habilidad para continuar más allá de este extremo resulta de un método para aproximarse al criterio de selección de movimientos, especificado por el método estándar del re-encadenamiento de camino, que busca la próxima solución que deja el menor número de movimientos restantes para alcanzar x''. Específicamente, sea $A(x)$ el conjunto de atributos de solución en x, y sea A_{drop} el conjunto de atributos de solución que se sacan por los movimientos ejecutados para alcanzar la solución actual $x'(i)$, es decir, los atributos que han servido como *DesdeAtributos* en estos movimientos. Entonces buscamos un movimiento en cada paso que maximice el número de $HaciaAtributos$ que pertenecen a $A(x'') - A(x'(i))$, y sujeto a ello que minimice el número de los que pertenecen a $A_{drop} - A(x'')$. Tal regla generalmente puede implementarse muy eficientemente, a través de estructuras de datos limitando el examen de movimientos a aquellos que contienen $HaciaAtributos$ de $A(x'') - A(x'(i))$ (o permitiendo que estos movimientos se examinen antes que otros).

Una vez que se alcanza $x'(r) = x"$, el proceso continúa modificando la regla

de elección de la siguiente forma. El criterio ahora selecciona un movimiento para maximizar el número de $HaciaAtributos$ fuera de A_{drop} menos el número de sus $HaciaAtributos$ que están en A_{drop}, y sujeto a esto minimizar el número de sus $DesdeAtributos$ que pertenecen a $A(x'')$. El camino entonces se detiene donde no quede elección que permita que el criterio de maximización sea positivo.

Para entornos que permitan elección de movimientos relativamente no restringida, este método produce una extensión más allá de x'' que introduce nuevos atributos, sin reincorporar ningún antiguo atributo hasta que no quede ningún movimiento que satisfaga esta condición. La habilidad para ir más allá de los extremos x' y x'' crea una forma de diversificación que no es accesible desde el camino que queda entre estos extremos. Al mismo tiempo, los puntos exteriores están influidos por la trayectoria que enlaza x' y x''.

Soluciones evaluadas, pero no visitadas. Las estrategias de intensificación y diversificación pueden beneficiarse del hecho de que un proceso de búsqueda genera información no sólo sobre las soluciones realmente visitadas, sino también acerca de soluciones adicionales evaluadas durante el examen de los movimientos no adoptados. Una manifestación de esto es explotada en referencia a las soluciones $x^*(i)$ en el método de re-encadenamiento de camino. Desde un punto de vista diferente, sea S^* un subconjunto de soluciones evaluadas, pero no visitadas (es decir, tomadas de la secuencia $x(1), ..., x(iteracionactual)$ cuyos elementos x dan valores $c(x)$ dentro de una banda de atracción elegida). Es relativamente fácil mantener un contador tal como $\#S^*(\text{a } x_j = p)$, que identifica el número de veces que $x_j = p$ es un $HaciaAtributo$ de un intento de movimiento que lleva a una solución de S^*. Tal contador puede diferenciarse aún más estipulando que el movimiento probado debe ser de mejora, y de alta calidad relativa a otros movimientos examinados en la misma iteración. Entonces a un atributo que alcanza una frecuencia relativamente alta sobre S^*, pero que tiene una baja frecuencia de residencia sobre las soluciones realmente visitadas, se le da un incentivo para incorporarlo a futuros movimientos, sirviendo simultáneamente para los objetivos de intensificación y diversificación. Lo reciente y lo frecuente interactúan en este método separando el incentivo si el atributo ha sido seleccionado en un movimiento reciente.

Penalizaciones e incentivos específicos de intervalo. Un ajuste útil de las ideas precedentes extiende la filosofía de la Aspiración por Dirección de Búsqueda y Aspiración por Admisibilidad Fuerte. Por estos criterios de aspiración, los movimientos de mejora se pueden escapar de la clasificación tabú bajo ciertas condiciones, pero con el resultado de rebajar su estado de tal manera que sean tratados como movimientos de mejora inferiores. Una extensión de esto preserva la distinción mejora/no-mejora cuando se introducen las penalizaciones e incentivos que no se pretende que sean preventivos. Para esta extensión, las evaluaciones vuelven a dividirse en intervalos de mejora y no mejora. Las penalizaciones y los incentivos se dan con alcance limitado, degradando o realzando las evaluaciones dentro de un

intervalo, pero sin alterar la relación entre las evaluaciones que caen en intervalos diferentes.

Los incentivos concedidos en base a la similitud de influencia se hacen sujetos a este desplazamiento restringido de las evaluaciones. Dado que un movimiento influyente usualmente no es de mejora en el entorno de un óptimo local, mantener la relación entre evaluaciones de diferentes intervalos implica que tales movimientos se seleccionarán sólo cuando no existan otros movimientos de mejora que los clasificados tabú. Pero los movimientos influyentes también tienen un efecto basado en lo reciente. Sólo la ejecución de un movimiento de alta influencia puede cancelar la clasificación tabú de un movimiento de menor influencia sobre una serie de iteraciones, por lo que debería reducir o cancelar el incentivo para seleccionar otros movimientos influyentes por una duración correspondiente.

Procedimientos de Listas de Candidatos. Anteriormente, ya se ha destacado la importancia de los procedimientos para aislar un conjunto de movimientos candidatos de un entorno grande, para evitar el gasto computacional de evaluar todo el entorno. Algunos procedimientos de este tipo han sido utilizados en métodos de optimización desde que el tema de la reducción de los esfuerzos computacionales se ha tomado en serio (desde al menos los años 50 y probablemente antes). Alguna de las formas más estratégicas de estos problemas vienen del campo de la optimización de redes [14]. En tales métodos, el subconjunto de movimientos se referencia mediante una lista que identifca sus elementos definitorios (tales como índices de variables, nodos y arcos), y por tanto estos métodos han adquirido el nombre de *estrategias de listas de candidatos*.

Una forma simple de estrategia de lista de candidatos es construir una lista simple de elementos muestreando el entorno al azar, y repetir el proceso si el resultado se estima inaceptable. Este es el fundamento de los métodos de Monte Carlo. Sin embargo, algunos estudios de optimización de redes, sugieren que los procedimientos basados en diseños más sistemáticos producen resultados superiores. Generalmente, éstos incluyen la descomposición del entorno en subconjuntos críticos, y el uso de una regla que asegure que los subconjuntos no examinados en una iteración se planifiquen para ser examinados en iteraciones siguientes. Para los subconjuntos apropiadamente determinados, los mejores resultados se obtienen seleccionando los movimientos de máxima calidad de estos subconjuntos, bien examinando explícitamente todas las alternativas o usando un umbral adaptativo para identificar tales movimientos.

Otra clase de estrategias de listas de candidatos examina periódicamente porciones más grandes del entorno, creando una lista maestra de algún número de las mejores alternativas encontradas. La lista maestra se consulta entonces para identificar movimientos (derivados o relacionados con los almacenados) para iteraciones adicionales hasta que un umbral de aceptabilidad dispara la creación de una nueva lista maestra.

Las estrategias de listas de candidatos implícitamente tienen una influencia

diversificante motivando que diferentes partes del espacio de entorno se examinen en diferentes iteraciones. Esto sugiere que debe beneficiarse de coordinar tales estrategias con otras estrategias de diversificación, un área que permanece abierta a la investigación. Las estrategias de listas de candidatos también son muy naturales para procesos de paralelización, donde se examinan en paralelo formas de descomposición de entornos a examinar de forma secuencial. Los movimientos pueden seleccionarse eligiendo el mejor candidato por varios procesos, o en su lugar cada proceso puede ejecutar sus propios movimientos preferidos, generando trayectorias de soluciones paralelas que son periódicamente coordinadas a un nivel superior. Estos últimos procedimientos se mantienen considerablemente prometedores.

Entornos compuestos. La identificación de un entorno efectivo para definir los movimientos desde una solución a otra puede ser extremadamente importante. Por ejemplo, un intento de resolver un problema de programación lineal eligiendo los movimientos que incrementan o decrementan variables del problema, frente a elegir movimientos que usan procesos de pivotaje o direcciones de búsqueda, obviamente puede provocar una diferencia sustancial en la calidad de la solución final obtenida. Las innovaciones que han hecho a la programación lineal una potente herramienta de optimización dependen significativamente del descubrimiento de entornos efectivos para hacer los movimientos.

Para aplicaciones combinatorias donde las posibilidades para crear entornos están ampliamente confinadas a varios procesos constructivos o destructivos, o a intercambios, mejoran frecuentemente los resultados combinando entornos para crear movimientos. Por ejemplo, en aplicaciones de secuenciación generalmente es preferible combinar entornos consistentes en movimientos de inserción y movimientos de intercambio, permitiendo considerar ambos tipos de movimientos en cada paso. Otra forma de combinar entornos es generar movimientos combinatorios, donde una secuencia de movimientos simples es tratada como un solo movimiento más complejo.

Un tipo especial de método para crear movimientos compuestos resulta de una sucesión de pasos en los que un elemento es asignado a un nuevo estado, con la consecuencia de *expulsión de* algún otro elemento de su estado actual. El elemento expulsado se asigna a su vez a un nuevo estado, expulsando a otro elemento, y así sucesivamente, creando una cadena de tales operaciones. Por ejemplo, tales procesos ocurren en un problema de secuenciación de tareas al mover una tarea a una nueva posición ocupada por otra tarea, expulsando esta tarea de su posición. La segunda tarea entonces se mueve a una nueva posición expulsando aún otra tarea, y así sucesivamente. Finalmente se acaba por insertar la última tarea entre dos tareas que son actualmente adyacentes. Este tipo de método llamado, estrategia de expulsiones en cadena, incluye la expulsión de enlaces entre elementos (tales como tareas) más que expulsar los elementos en sí, y también se aplica a elementos agregados y a enlaces. Las estrategias de

expulsiones en cadena tienen aplicaciones útiles en problemas de muchos tipos, particularmente en conexión con planificación, rutas, clasificación y partición [1], [10].

Oscilación Estratégica. El método de oscilación estratégica está estrechamente relacionado con los orígenes de la búsqueda tabú, y proporciona una técnica efectiva entre intensificación y diversificación para medio a largo plazo. La oscilación estratégica opera moviendo hasta chocar con una frontera, representada por la factibilidad o una etapa de construcción, que normalmente representaría un punto donde el método se pararía. En vez de parar, sin embargo, la definición de entorno se extiende o el criterio de evaluación para seleccionar movimientos se modifica para permitir que la frontera se cruce. El método entonces continúa por una profundidad especificada más allá de la frontera y luego vuelve. En este punto se vuelve a aproximar a la frontera y se cruza, esta vez en dirección opuesta, procediendo a un nuevo punto de giro. El proceso de acercarse repetidamente y cruzar la frontera desde diferentes direcciones crea una forma de oscilación que da al método su nombre. El control sobre esta oscilación se establece generando evaluaciones modificadas y reglas de movimiento, dependiendo de la región en la que se está actualmente navegando y de la dirección de búsqueda. La posibilidad de recorrer de nuevo una trayectoria anterior se evita con los mecanismos tabú estándares.

Un ejemplo de este método ocurre para el problema de la mochila multidimensional, donde los valores de las variables 0-1 se cambian de 0 a 1 hasta que se alcanza la frontera de factibilidad. El método entonces continúa dentro de la región no factible usando el mismo tipo de cambios, pero con un evaluador modificado. Después de un número seleccionado de pasos, la dirección se invierte cambiando variables de 1 a 0. El criterio de evaluación conduce hacia la mejor variación (o la de menor empeoramiento) de acuerdo a si el movimiento es de más a menos o de menos a más factible (o no factible), y se acompaña por las restricciones asociadas sobre los cambios admisibles de valores en las variables. Una implementación de tal método de [2], [3] ha generado soluciones particulares de alta calidad para el problema de la mochila multidimensional.

Un tipo algo diferente de aplicación ocurre para el problema de encontrar un árbol generador óptimo sujeto a restricciones de desigualdad en un conjunto de aristas ponderadas. Un tipo de método de oscilación estratégica para este problema resulta de un proceso constructivo de añadir aristas a un árbol que crece hasta que es generador, y entonces continúa añadiendo aristas para cruzar la frontera definida por la construcción del árbol. Un grafo diferente se obtiene cuando la solución actual no es un árbol, y por tanto se requiere un entorno diferente, produciendo reglas modificadas de selección de movimientos. Las reglas cambian otra vez para proceder en la dirección contraria, quitando aristas hasta conseguir otra vez un árbol. En tales problemas, el esfuerzo requerido por diferentes reglas puede hacer preferible cruzar la frontera con diferente profundidad por

diferentes sitios. Una opción es aproximarse y retirarse de la frontera mientras permanece a un solo lado, sin cruzar (es decir, eligiendo cruzar con profundidad 0). En este ejemplo, se pueden considerar otros tipos de frontera, derivados de las restricciones de desigualdad.

El uso de oscilación estratégica en aplicaciones que alternan procesos constructivos y destructivos puede acompañarse de movimientos de intercambio que mantienen la construcción a un determinado nivel. Un *principio de optimalidad aproximada*, que establece aproximadamente que buenas construcciones a un nivel son más probables de estar cerca de buenas construcciones a otro nivel, motiva una estrategia de aplicar intercambios a distintos niveles, a cada lado de una estructura blanco o diana tal como el árbol generador, para obtener construcciones refinadas antes de pasar a niveles adyacentes.

Finalmente, remarcamos que la frontera incorporada en la oscilación estratégica no necesita definirse en términos de factibilidad o estructura, pero puede definirse en términos de una región donde la búsqueda parece gravitar. La oscilación entonces consiste en obligar a la búsqueda a salir de esta región y permitirle volver.

Apéndice. Elementos a considerar al implementar una Búsqueda Tabú.

Durante el proceso de diseño de un método de búsqueda tabú para una aplicación particular, puede resultar de utilidad plantearse algunas de las cuestiones que se listan en este apéndice. Durante la presentación de estas cuestiones haremos referencia, como bibliografía adicional al presente capítulo, a diversas secciones del libro "Tabu Search" (TS) [8], escrito por los profesores Fred Glover y Manuel Laguna en el año 1997. Para hacer alusión a sus capítulos, por ejemplo al capítulo 3, usaremos la notación TS(3). De la misma forma, para hacer alusión, por ejemplo a la sección 2 del capítulo 3, mostraremos TS(3.2).

Independientemente del método de búsqueda utilizado para resolver un problema particular, hay cuatro elementos comunes a todos ellos: (i) la representación de la solución, la cual permite generar estructuras de entorno adecuadas para el problema, (ii) un objetivo, (iii) una función de evaluación, y (iv) un mecanismo de movimiento, proporcionado por el método usado; búsqueda tabú en nuestro caso. En lo que sigue, mostramos la lista de cuestiones que es conveniente plantearse antes de comenzar el diseño de una búsqueda tabú.

1. *El tipo de entorno usado.* ¿Hay más de una posible estructura de entorno que pueda ser potencialmente relevante? Cuando la respuesta a la pregunta anterior es positiva, ¿pueden combinarse o alternarse los entornos de forma conveniente? ¿Es posible (y deseable) usar entornos que hayan surgido de la resolución de forma óptima de problemas relacionados o ralajados? ¿Puede un entorno asociarse más libremente a la solución de un problema auxiliar

o relajado? ¿En qué medida puede la estructura del problema actual ser explotada por la definición del entorno? (Ver TS(10.5), 351-352)

2. *Indentificar los atributos de las soluciones.* Estos atributos, que se derivan de los movimientos determinados por las estructuras de entorno seleccionadas, en ocasiones toman la forma de "variables del problema". Pueden ser acciones, decisiones, elementos asignados a determinadas localizaciones o períodos de tiempo, cantidades de artículos comprados o adquiridos, y así sucesivamente.

Referente a la memoria a corto plazo basada en lo reciente

3. *Definición de los estados tabú relacionados con los atributos identificados en el punto 2.* Para aquellos movimientos que involucran el cambio de más de un atributo al mismo tiempo, cada atributo componente tiene asignado un estado tabú (por ejemplo, tal como hemos mencionado anteriormente en este capítulo, *tabú-activo* o *tabú-inactivo*). El número de iteraciones usadas para determinar la duración de un estado tabú-activo puede despender del atributo considerado, tal como se ha explicitado en el ejemplo de los dos últimos párrafos de la sección 3.4 de este capítulo.

4. La naturaleza del período tabú para los atributos de las soluciones (fija o dinámica), y la regla de decisión para inicializar o actualizar el período (aleatoria o sistemática) explicados en la sección 3.4 son otros dos de los elementos fundamentales en el diseño de la búsqueda tabú. El "Método de Ciclo Tabú", descrito en TS(7.3.3) 241-244, constituye un ejemplo de estrategia dinámica, cuya eficacia ha sido demostrada recientemente.

5. Determinar el uso de los criterios de aspiración usados para eliminar estados tabú, explicados en la sección 3.5 de este capítulo. Para obtener información más completa sobre los mismos, se recomienda revisar TS(2.6) 50-54.

Referente a la memoria a largo plazo

6. *Las estructuras y los usos hechos de la memoria basada en frecuencia.* Por ejemplo, ¿qué tipos de soluciones o movimientos proporcionan la base para aplicar esta memoria? ¿Almacena el proceso la frecuencia con la que aparecen determinados atributos en soluciones o movimientos élite? ¿Constituye la memoria de frecuencia de transición, tal como aquella que cuenta únicamente el número de veces que un atributo fue añadido para crear una solución élite, o es una frecuencia de residencia, que cuenta el número de soluciones (o soluciones élite) en las que ha estado presente un atributo? De forma similar, ¿hace uso el método de frecuencias relativas a soluciones

mediocres o malas? Por último, ¿tiene la memoria basada en frecuencia un uso a corto plazo al igual que en su uso a largo plazo? Para obtener más información que permite dar respuesta a estas preguntas, véase TS(4.1) 94-94 y TS(4.7) 117-121.

7. *Estrategias usadas para la intensificación.* ¿Realiza el método un almacenamiento explícito de las soluciones élite y busca periódicamente explorar otras soluciones cercanas a éstas, o confía sobre todo en reforzar la elección de atributos que tienen una alta frecuencia por aparecer en soluciones buenas, tal como se determinaba en el punto 6? Para encontrar la respuesta a esta preguntas se puede consultar además TS(4.2) 96-98.

8. *Estrategias usadas para la diversificación.* ¿Usa el método una estrategia de multi-arranque para obtener diversificación, o usa periódicamente movimientos que guían la búsqueda hacia zonas alejadas de las regiones ya visitadas? Véase TS(4.3) 98-102 y las primeras partes de TS(5).

9. Uso de algunas estrategias fundamentales tales como oscilación estratégica, re-encadenamiento de caminos, proyección de memoria adaptativa, etc. ¿Se usa alguno de estos métodos como espina dorsal de la búsqueda? ¿Son invocados periódicamente en un papel combinado de intensificación/diversificación?

10. ¿Qué estructuras de datos pueden usarse para facillitar los elementos anteriores, incluyendo estructuras que almacenen información suficiente para permitir la rápida actualización de las evaluaciones de los movimientos en vez de recalcular las soluciones completamente? (TS(3.1) 59-61.)

11. Las estrategias de listas de candidatos, que porporcionan un modo para enfocar las reglas de decisión y reducir la computación global, constituyen también un elemento fundamental de la búsqueda tabú (TS(3.2) 61-67). Véase también el ejemplo completo de la sección 2.3 de este capítulo.

Nota: En algunos problemas relacionados con la planificación, los entornos que parecen ser los más fáciles y naturales son aquellos basados en procesos constructivos y destructivos, construyendo un plan paso a paso. En este caso, a menudo es útil un enfoque de búsqueda tabú multi-arranque para generar una nueva solución en cada paso. Sin embargo, en muchas aplicaciones en las que un procedimiento constructivo o destructivo puede que no proporcione el enfoque más simple, pudiera resultar valioso el uso de otras estructuras de entorno, siguiendo las consideraciones del punto 1. (En los artículos 294 y 302 que aparecen en "Publications" de la página web http://spot.colorado.edu/∼glover, se encuentran referencias sobre métodos de búsqueda tabú multi-arranque).

5 Bibliografía

[1] Dorndorf, U. y Pesch, E. (1994) "Fast Clustering Algorithms", *ORSA Journal on Computing*. 6:2, 141-153.

[2] Freville, A. y Plateau, G. (1986) "Heuristics and Reduction Methods for Multiple Constraint 0-1 Linear Programming Problems", *European Journal of Operational Research*, 24, 206-215.

[3] Freville, A. y Plateau, G. (1990) "Hard 0-1 multiknapsack test problems for size reduction methods", *Invesstigacion Operativa*, 1, 251-270.

[4] Glover, F. (1977) "Heuristics for Integer Programming Using Surrogate Constraints", *Decision Science*. 8, 156- 166.

[5] Glover, F. (1986) "Future Paths for Integer Programming and Links to Artificial Intelligence", *Computers and Operations Research*. 5. 533-549.

[6] Glover, F. y Laguna, M. (1993). Tabu Search. *Modern Heuristic Techniques for Combinatorial Problems*, C. Reeves, ed., Blackwell Scientific Publishing, pp. 71-140.

[7] Glover, F. (1996) "Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problems", *Discrete Applied Mathematics*, 65, 223-253.

[8] Glover, F. and Laguna, M. (1997). *Tabu Search*, Kluwer Academic Publishers.

[9] Glover, F., Laguna, M. and Martí R. (2006) *Principles of Tabu Search*. To appear in *Approximation Algorithms and Metaheuristics*, Chapman & Hall/CRC.

[10] Mulvey, J. (1978) "Pivot strategies for primal simplex network codes", *Journal of the Association for Computing Machinery*, 25, 266-270.

Segunda parte

Metaheurísticos

GRASP: Procedimientos de búsqueda miopes, aleatorizados y adaptativos*

José Luis González Velarde
Tecnológico de Monterrey
Monterrey, NL 64849 México
gonzalez.velarde@itesm.mx

1 Introducción

Consideremos un problema de optimización combinatoria definido por un conjunto base finito E = {1, ..., n}, un conjunto de soluciones factibles $F \subseteq 2^E$, y una función objetivo $f : 2^E \longrightarrow \mathbb{R}$. En la versión de minimización, buscamos una solución óptima $S^* \in F$ tal que $f(S^*) \leq f(S)$; $\forall S \in F$. El conjunto base E, la función de coste f, así como el conjunto de soluciones factibles F se definen para cada problema específico. Por ejemplo, en el caso del problema del agente viajero, el conjunto base E consta de todas las aristas que conectan las ciudades a ser visitadas, $f(S)$ es la suma de los costes de todas las aristas $e \in S$, y F está formado por todos los subconjuntos de aristas que determinan un ciclo Hamiltoniano.

Un procedimiento de búsqueda miope aleatorizado y adaptativo (GRASP por sus siglas en inglés) [42, 43] es una metaheurística para encontrar soluciones aproximadas (i.e. sub-óptimas de buena calidad, pero no necesariamente óptimas) a problemas de optimización combinatoria. Se basa en la premisa de que soluciones iniciales diversas y de buena calidad juegan un papel importante en el éxito de métodos locales de búsqueda.

*El autor agradece el apoyo recibido por el Tecnológico de Monterrey a través de la cátedra CAT025 para la preparación de este manuscrito.

Un GRASP es un método multi-arranque, en el cual cada iteración GRASP consiste en la construcción de una solución miope aleatorizada seguida de una búsqueda local usando la solución construida como el punto inicial de la búsqueda local. Este procedimiento se repite varias veces y la mejor solución encontrada sobre todas las iteraciones GRASP se devuelve como la solución aproximada. El seudo-código en la Figura 1 ilustra un GRASP básico para minimización.

```
        procedure GRASP
f* ← ∞
for i ≤ imáx do
        x ← GreedyRandomized();
        x ← LocalSearch(x);
        if f(x) < f* then
                f* ← f(x);
                x* ← x;
        end if
        i ← i + 1
end for
return x*
```

Figura 1.

En este capítulo, nos centraremos primero en las dos componentes más importantes de GRASP. A saber, construcción y búsqueda local. Después examinaremos cómo el reencadenamiento de trayectorias puede ser usado en GRASP como un mecanismo de memoria e intensificación. El trabajo termina con una lista parcial de aplicaciones exitosas de GRASP.

Reseñas recientes de GRASP pueden encontrarse en [102, 93], una extensa bibliografía comentada está en [50] y una actualización en el URL

http://graspheuristic.org/annotated.

2 Construcciones GRASP

En esta sección describimos varios mecanismos de construcciones miopes aleatorizadas. Estos procedimientos mezclan miopía con aleatorización de diferentes formas. Todos los mecanismos de construcción considerados construyen una solución incorporando un elemento a la vez. En cada paso del proceso de construcción, se tiene a la mano una solución parcial. Un elemento que pueda seleccionarse como parte de una solución parcial se llama elemento candidato. Consideremos un problema de cubrimiento de conjuntos, donde se tiene una matriz

$A = [a_{ij}]$ de ceros y unos, un coste c_j para cada columna j, y se quiere determinar un conjunto J de columnas con el menor coste total $\sum_{j \in J} c_j$ tal que para cada renglón i, al menos una columna j del conjunto tenga una entrada $a_{ij} = 1$. En este problema, una solución parcial es un conjunto de columnas que no necesariamente forman un cubrimiento. Cualquier columna que no se haya seleccionado previamente es un elemento candidato. El conjunto solución J se construye incorporando un elemento (columna) a la vez hasta que el conjunto J sea un cubrimiento.

Para determinar qué elemento candidato seleccionar enseguida para incluirse en la solución, generalmente se hace uso de una función miope. Una función miope mide la contribución local de cada elemento a la solución parcial. En el caso del cubrimiento de conjuntos, una función miope plausible es la razón entre el número p_j de filas sin cubrir, que quedarían cubiertas si la columna j se elige y la contribución c_j al coste total de elegir la columna j para la solución, esto es p_j/c_j. La elección miope sería agregar la columna con el mejor valor de la función miope.

Procedure Construcción-C
Input: k, E, $c(\_)$;
$x \leftarrow \emptyset$;
$C \leftarrow E$;
while $C \neq \emptyset$ **do**
Calcular el costo miope $c(e)$; $\forall e \in C$;
RCL $\leftarrow \{k$ elementos $e \in C$ con el menor $c(e)\}$;
Seleccionar un elemento s de RCL al azar;
$x \leftarrow x \cup \{s\}$;
Actualizar el conjunto candidato C;
end while
return x;

Figura 2

Existen varias formas posibles de introducir aleatoridad a este procedimiento. Una de las primeras ideas fue el uso de una lista restringida de candidatos (RCL) [42]. Tal lista contiene un conjunto de elementos candidatos con los mejores valores de la función miope. El siguiente candidato a ser agregado a la solución se selecciona al azar de la lista restringida de candidatos. Dicha lista puede consistir de un número fijo de elementos (restricción por cardinalidad) o elementos con los valores de la función miope dentro de un rango dado. La Figura 2 muestra un seudo-código para un procedimiento de construcción GRASP basado en restricción por cardinalidad. Por ejemplo, denotemos por c^* y c_*, respectivamente, los valores mayor y menor de la función miope para los elementos candidatos, y

sea α un número real tal que $0 \leq \alpha \leq 1$. En una lista restringida de candidatos basada en el valor, la RCL consiste en todos los elementos candidatos e cuyo valor de función miope $c(e)$ es tal que $c(e) \leq c_* + \alpha(c^* - c_*)$. Nótese que si $\alpha = 0$, entonces este esquema de selección es un algoritmo miope, mientras que si $\alpha = 1$, entonces es totalmente aleatorio. La Figura 3 muestra un seudo- código para un procedimiento de construcción GRASP basado en el valor. Más tarde será discutida la forma de determinar valores para α.

procedure Construcción-V
Input: $\alpha, E, c(\_)$;
$x \leftarrow \emptyset$;
$C \leftarrow E$;
while $C \neq \emptyset$ **do**
Calcular el costo miope $c(e)$; $\forall e \in C$;
$c_m = \min\{c(e) \mid e \in C\}$;
$c^M = \max\{c(e) \mid e \in C\}$;
RCL $\leftarrow \{e \in C \mid c(e) \leq c_m + \alpha(c^M - c_m)\}$;
Seleccionar un elemento s de RCL al azar;
$x \leftarrow x \cup \{s\}$;
Actualizar el conjunto candidato C;
end while
return x;

Figura 3

Se puede también mezclar una construcción al azar con una construcción miope de la siguiente manera. Elegir secuencialmente un conjunto parcial de elementos candidato al azar y después completar la solución usando un algoritmo miope [95]. La Figura 4 muestra un seudo-código para tal procedimiento de construcción.

procedure Construcción-RG
input: k, E, c( );
$x \leftarrow \emptyset$;
$C \leftarrow E$ **for** $i = 1, 2, ..., k$ **do**
  if $C \neq \emptyset$ **then**
    Elegir el elemento e al azar de C;
    $x \leftarrow x \cup \{e\}$;
    Actualizar el conjunto de candidatos C;
  end if
end for
while $C \neq \emptyset$ **do**
  Calcular el costo miope $c(e)$; $\forall e \in C$
  $e_+ \leftarrow \text{argmin}\{c(e) \mid e \in C\}$;
  $x \leftarrow x \cup \{e_+\}$;
  Actualizar el conjunto de candidatos C;
  Calcular el costo miope $c(e)$;$\forall e \in C$;
end while
return x;

Figura 4

Otro enfoque es mediante perturbación de costes. Aquí, los datos de costes se perturban aleatoriamente y se aplica un algoritmo miope [27]. La Figura 5 muestra un seudo-código para este procedimiento de construcción.

procedure Construcción-PG
input: $E, c(\_)$;
$x \leftarrow \emptyset$;
$C \leftarrow E$;
Perturbar aleatoriamente los datos del problema;
while $C \leftarrow E$; **do**
  Calcular el costo miope perturbado $\underline{c}(e)$; $\forall e \in C$;
  $e^* \leftarrow \text{argmin}\{c(e) \mid e \in C\}$;
  $x \leftarrow x \cup \{e^*\}$;
  Actualizar el conjunto de candidatos C;
end while
return x;

Figura 5

Un ejemplo final de un procedimiento de construcción de GRASP es una

variación sobre el enfoque de RCL basado en el valor. En este procedimiento, llamado función de sesgo [25], en vez de seleccionar el elemento de la RCL al azar con iguales probabilidades asignadas a cada elemento, se asignan diferentes probabilidades, favoreciendo elementos bien evaluados. Los elementos de la RCL se ordenan de acuerdo a los valores de la función miope.

La probabilidad $\pi(r(e))$ de seleccionar el elemento e es

$$\pi(r(e)) = \frac{sesgo(r(e))}{\sum_{e' \in RCL} sesgo(r(e'))},$$

donde $r(e)$ es la posición del elemento e en la RCL. Se han propuesto varias alternativas para asignar sesgos a los elementos. Por ejemplo,

- sesgo aleatorio: $sesgo(r) = 1$;
- sesgo lineal: $sesgo(r) = 1/r$;
- sesgo exponencial: $sesgo(r) = e^{-r}$.

En la siguiente sección, discutimos cómo determinar el valor de α para usarse en los esquemas basados en RCL anteriormente discutidos. Recordemos que si $\alpha = 0$, entonces estos esquemas de selección se convierten en un algoritmo miope, mientras que si $\alpha = 1$, son totalmente aleatorios.

3 Búsqueda local

Un algoritmo de búsqueda local explora repetidamente la vecindad de una solución en busca de una mejor solución. Cuando no se encuentra una solución que mejora la actual, se dice que la solución es localmente óptima. La Figura 6 muestra un seudo-código para un procedimiento de búsqueda local.

procedure BúsquedaLocal
input: $x_0, N(\_), f(\_)$;
$x \leftarrow x_0$;
while x no es localmente óptimo con respecto a $N(x)$ **do**
    Sea $y \in N(x)$ tal que $f(y) < f(x)$;
    $x \leftarrow y$;
end while
return x;

Figura 6

La búsqueda local juega un papel importante en GRASP ya que sirve para buscar soluciones localmente óptimas en regiones prometedoras del espacio de

soluciones. Esta es la diferenciación de GRASP con respecto del algoritmo semi-miope de Hart y Shogan [55]. Por definición su desempeño nunca será peor que semi-miope, y casi siempre producirá mejores soluciones en menos tiempo.

Aunque los algoritmos miopes pueden producir buenas soluciones razonables, su principal desventaja como generador de soluciones iniciales para búsquedas locales es su falta de diversidad. Aplicando repetidamente un algoritmo miope, una sola o muy pocas soluciones pueden generarse. Por otra parte, un algoritmo totalmente aleatorio produce una gran cantidad de soluciones diversas. Sin embargo, la calidad de estas soluciones generalmente es muy pobre y usarlas como soluciones iniciales para búsquedas locales generalmente conduce a una convergencia lenta hacia un mínimo local.

Para beneficiarse de la convergencia rápida del algoritmo miope y de la gran diversidad del algoritmo aleatorio, se acostumbra usar un valor de α estrictamente contenido en el interior del rango $[0; 1]$. Ya que no se conoce a priori qué valor usar, se han propuesto diferentes esquemas. La primera referencia en la literatura en la cual se propone una variación del parámetro α fue [66]. Los autores proponen un valor inicial del parámetro, $\alpha = 1$, con este valor se efectúan las construcciones, una vez que han transcurrido un cierto número de iteraciones sin que se haya construido una solución mejor, este valor se disminuye en una cantidad $\Delta\alpha$, esto es $\alpha = \alpha - \Delta\alpha$. Esto se repite mientras el parámetro α, no sea negativo. Otra estrategia razonable es seleccionar al azar un valor diferente en cada iteración GRASP. Esto puede hacerse usando una probabilidad uniforme [92] o usando el esquema de GRASP reactivo [88].

En el esquema de GRASP reactivo, sea $\Psi = \{\alpha_1, \ldots, \alpha_m\}$ el conjunto de valores posibles para α. Las probabilidades asociadas con la elección de cada valor se fijan todas inicialmente iguales a $p_i = 1/m;\ i = 1; ::: ; m$. Más aún, sea z^* el valor de la solución incumbente, esto es, la mejor solución encontrada hasta el momento, y sea A_i el valor promedio de todas las soluciones halladas usando $\alpha = \alpha_i\ i = 1; ::: ; m$. Las probabilidades de selección se reevalúan periódicamente tomando $p : i = q_i / \sum_{j=1,\ldots,m} q_j$, con $q_i = z^*/A_i$ para $i = 1; ::: ; m$. El valor de q_i será mayor para valores de $\alpha = \alpha_i$ que produzcan las mejores soluciones en promedio. Mayores valores de q_i corresponden a valores del parámetro α más adecuados. Las probabilidades asociadas con estos valores más apropiados se incrementarán cuando sean reevaluadas.

En el contexto de GRASP, se han usado esquemas de búsqueda local más elaborados. Por ejemplo, búsqueda tabú [66, 35, 1, 101], recocido simulado [70], vecindades variables [28, 49], y vecindades extendidas [3].

4 Estructuras de memoria en GRASP

Quizás una de las principales desventajas del planteamiento original de GRASP es su falta de estructuras de memoria. Las iteraciones de GRASP son independientes y no utilizan las observaciones hechas durante iteraciones previas. Una consecuencia de esto es el hecho de que una solución previamente construida puede aparecer nuevamente, ya que esta situación es equivalente a un muestreo con reemplazo, invirtiendo tiempo de cómputo en construir soluciones repetidas.

Un remedio ha sido sugerido por Fleurent y Glover [51] quienes usan un diseño de memoria adaptativo tal como se propone en búsqueda tabú con el fin de retener y analizar características de ciertas soluciones seleccionadas y almacenadas en un conjunto élite S, y proporcionar una base para mejorar las ejecuciones futuras dentro del proceso constructivo.

Para esto definen una función de evaluación $E(e) = F(valor(e), intensidad(e))$, para cada candidato en la lista C. F es una función monótona no decreciente en sus argumentos, donde $valor(e)$ está asociada con la función objetivo. Mayores valores de esta función corresponden a mejores elecciones (así, en un problema de minimización, $valor(e)$ se incrementa si el cambio en el coste disminuye). Mientras que $intensidad(e)$, se vuelve más grande cuando e aparece con más frecuencia en los mejores miembros del conjunto élite S.

Esta función es utilizada entonces para determinar las probabilidades de elección de cada miembro de la lista C, $p(e) = E(e)/\sum_{e' \in C} E(e')$. La función de evaluación generalmente tiene la forma $E(e) = \lambda valor(e) + intensidad(e))$. Valores mayores de λ dan más énfasis a valor, que a intensidad, esto es deseable al inicio de la búsqueda, ya que no se cuenta con información suficiente para que el factor intensidad pueda ser significativo. A medida que avanza la búsqueda y se tiene información dentro del conjunto de soluciones élite, el valor de λ puede disminuirse. Ya que la intensificación de alguna manera enfatiza la calidad de las soluciones a costa de la aleatorización, generalmente se acompaña con una componente de diversificación que periódicamente conduce la búsqueda hacia la exploración de diferentes regiones del espacio de búsqueda. En este caso, la diversificación se puede lograr incrementando λ, lo cual se hace cuando la diversidad de las soluciones generadas es muy baja.

Otra alternativa es el uso del reencadenamiento de trayectorias (path relinking) con GRASP. El reencadenamiento de trayectorias fue propuesto originalmente por Glover [53] como una forma de explorar las trayectorias entre soluciones élite obtenidas por búsqueda tabú o búsqueda dispersa. Usando una o más soluciones élite, se exploran las trayectorias en el espacio de soluciones que conducen a otras soluciones élite para buscar mejores soluciones. Para generar trayectorias, los movimientos se seleccionan para introducir atributos en la solución actual que estén presentes en la solución élite guía.

El reencadenamiento de trayectorias en el contexto de GRASP fue introducido por Laguna y Martí [70]. Desde entonces han aparecido numerosas extensiones, mejoras, y aplicaciones exitosas [5, 27, 94, 97, 95, 49]. Ha sido usado como un esquema de intensificación, en el que las soluciones generadas en cada iteración de GRASP se reencadenan con una o más soluciones de un conjunto élite de soluciones, o como en una fase de post-optimización, donde se reencadenan pares de soluciones del conjunto élite.

Consideremos dos soluciones x_s y x_t en las cuales queremos aplicar reencadenamiento de trayectorias desde x_s hacia x_t. La Figura 7 ilustra el procedimiento de reencadenamiento de trayectorias mediante su seudo-código. El procedimiento se inicia calculando la diferencia simétrica $\Delta(x_s; x_t)$ entre las dos soluciones, i.e. el conjunto de movimientos necesarios para alcanzar x_t desde x_s. Se genera entonces una trayectoria de soluciones encadenando a x_s con x_t. El algoritmo devuelve la mejor solución encontrada en esta trayectoria. En cada paso, el procedimiento examina todos los movimientos $m \in \Delta(x; x_t)$ desde la solución actual x y elige aquel que resulta en la solución menos costosa, i.e. aquel que minimiza $f(x \oplus m)$, donde $x \oplus m$ es la solución resultante de aplicar el movimiento m a la solución x. Se efectúa el mejor movimiento m^* produciendo $x \oplus m^*$ y el movimiento m^* se elimina de la diferencia simétrica de $\Delta(x \oplus m; x_t)$. En caso necesario, la mejor solución x^* se actualiza. El procedimiento termina cuando se alcanza x_t, i.e. cuando $\Delta(x; x_t) = \emptyset$.

procedure PR
input: x_s, x_t;
Calcular la diferencia simétrica $\Delta(x_s, x_t)$;
$x \leftarrow x_s$;
$f^* \leftarrow \text{mín}\{f(x_s), f(x_t)\}$;
$x^* \leftarrow \text{argmín}\{f(x_s), f(x_t)\}$;
while $\Delta(x_s, x_t) \neq \emptyset$; **do**
    $m^* \leftarrow \text{argmin}\{f(x \oplus m); \forall m \in \Delta(x, x_t)\}$;
    $\Delta(x \oplus m^*, x_t) \leftarrow \Delta(x, x_t) \setminus \{m^*\}$;
    $x \leftarrow x \oplus m^*$;
    if $f(x) < f^*$ **then**
        $f^* \leftarrow f(x)$;
        $x^* \leftarrow x$;
    end if
end while
return x^*;

Figura 7

La figura 8 ilustra el reencadenamiento de trayectorias.En el grafo de esta

figura, los nodos corresponden a soluciones, y los arcos corresponden a movimientos que permiten que una solución se alcance a partir de otra. Supóngase que dos soluciones, A y D, se reencadenan. Sea A la solución inicial y D la solución meta, y supóngase que la diferencia simétrica $\Delta(A; D) = 3$. Existen tres posibles movimientos partiendo de A. Se elige el mejor movimiento, el cual produce la solución B. En este punto, la diferencia simétrica $\Delta(B; D) = 2$ y por lo tanto existen dos movimientos posibles. De nuevo, se elige el mejor movimiento, el cual produce la solución C. Finalmente, en este punto hay un solo movimiento posible, el cual conduce a la solución meta D. Este esquema de reencadenamiento de trayectorias produjo una "trayectoria" $A \rightarrow B \rightarrow C \rightarrow D$ la cual puede ahora ser evaluada.

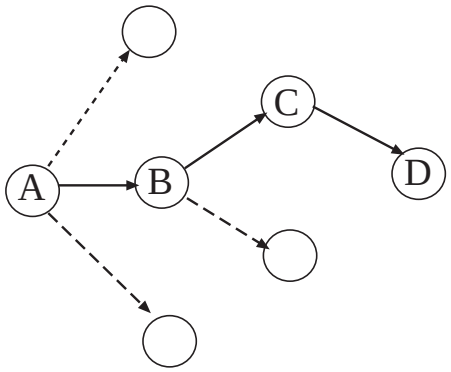


Figura 8

El reencadenamiento de trayectorias mantiene un conjunto P de soluciones élite halladas durante la optimización [51]. Las primeras $|P|$ soluciones distintas que se encuentran se insertan en el conjunto élite. Después de eso, una solución candidato x^* se agrega a P si su costo es menor que el costo de todas las soluciones del conjunto élite, o si su costo es mayor que el mejor, pero menor que la peor solución élite y es suficientemente diferente de todas las soluciones del conjunto élite. Si se acepta su entrada al conjunto élite, la nueva solución reemplaza a la solución más similar a ella entre el conjunto de soluciones élite con un costo peor que ella [95]. El conjunto élite puede ser renovado periódicamente [4] si no se observan cambios en el conjunto élite durante un número especificado de iteraciones GRASP. Una forma de hacer esto es fijar en infinito los valores de la función objetivo de la peor mitad del conjunto élite. De esta forma se crearán nuevas soluciones del conjunto élite.

Se han propuesto varios esquemas alternativos para el reencadenamiento de trayectorias. Ya que este procedimiento puede ser demandante en recursos computacionales, no necesita ser aplicado después de cada iteración GRASP. Es más

conveniente hacerlo periódicamente. Usualmente las trayectorias desde x_s hasta x_t y desde x_t hasta x_s son diferentes y ambas pueden explorarse. Ya que las trayectorias pueden ser largas, no es necesario seguir la trayectoria completa. Se puede restringir siguiendo una trayectoria truncada que inicie en x_s y otra que inicie en x_t.

5 GRASP Continuo

Recientemente Hirsch et al [57, 58, 59, 60], introdujeron un método de optimización global el cual extiende GRASP del dominio de la optimización discreta al de la optimización global continua. El dominio de la función se supone inmerso en un hiper-rectángulo de dimensión n, donde n es el número de variables. En el inicio de cada iteración GRASP, se genera de forma aleatoria una solución x dentro del rectángulo. La fase de construcción se inicia con esta solución dejando en libertad de variar todas sus coordenadas, en cada una de las coordenadas libres i se lleva a cabo una búsqueda lineal manteniendo las otras $n-1$ coordenadas de x en sus valores actuales. El valor de z_i de la i-ésima coordenada, que minimice el valor de la función objetivo, así como el valor de la función objetivo g_i se almacenan. Una vez que se ha llevado a cabo para cada una de las coordenadas libres la búsqueda local, se forma una lista restringida de candidatos (LRC) la cual contiene la coordenadas libres i cuyos valores g_i son menores o iguales a $\alpha\max + (1-\alpha)\min$, donde max y min son, respectivamente los valores máximo y mínimo de g_i sobre todas las coordenadas libres de x, y $\alpha \in [0;1]$. De la LRC se elige una coordenada al azar, digamos $j \in$ LRC, y x_j se fija a z_j, quedando $n-1$ variables libres. Eligiendo una coordenada de esta manera se asegura la aleatoreidad en la fase de construcción. Se continúa con el procedimiento anteriormente descrito hasta que todas las n coordenadas de x se hayan fijado. En este punto se ha obtenido x de la fase de construcción. Para la fase de post-procesamiento la cual consiste en una búsqueda local, iniciando desde un punto $x \in \mathbb{R}^n$, el algoritmo genera un conjunto de direcciones y determina en qué dirección, si es que hay una, mejora la función objetivo. Es fácil ver que existen $3^n - 1$ direcciones posibles y aun para valores moderados de n este número puede ser muy grande. Es por esto que se fija un cierto número máximo de direcciones a explorar, Ndir, y se construyen al azar este máximo número de direcciones. Una vez construida la dirección d, se genera el punto de prueba $x^* + hd$ donde h es un parámetro de discretización. Si el punto de prueba x es factible y es mejor que x^*, entonces a x^* se le asigna el valor de x y el proceso vuelve a comenzar con x^* como la solución inicial. Es importante notar que el conjunto de direcciones puede cambiar cada vez durante el proceso, así como el orden en el cual estas direcciones se consideran. Una vez que se encuentra un punto con $f(x^*) \leq f(x^* + hd)$ para cada una de las Ndir d elegidas, se declara a x^* localmente óptima y el procedimiento regresa

esta solución. Esto corresponde a una iteración de construcción de GRASP, el procedimiento se repite *maxiter* veces y se regresa la mejor de todas las soluciones construidas.

6 Aplicaciones

La primera aplicación de GRASP fue a cubrimientos de conjuntos [42] en 1989, y, a partir de entonces, ha sido aplicado a un amplio rango de tipos de problemas. Referimos al lector a Festa y Resende [50] y al URL

http://graspheuristic.org/annotated

para una extensa bibliografía comentada de GRASP. Concluimos este capítulo con una lista parcial de aplicaciones de GRASP, mostrando su amplia aplicabilidad.

- enrutamiento [11, 14, 19, 29, 64];
- lógica [36, 86, 90, 91];
- cubrimiento y partición [11, 12, 42, 52, 54];
- localización [1, 35, 70, 102, 103];
- árbol mínimo de Steiner [28, 75, 76, 77, 97];
- optimización en grafos [2, 44, 65, 84, 89, 93, 96, 48, 65, 32];
- asignación [41, 51, 66, 70, 78, 81, 85, 88, 87, 100, 56];
- horarios, programación, y manufactura [16, 17, 18, 21, 33, 37, 38, 39, 40, 45, 46, 66, 98, 99, 105, 6];
- transporte [11, 38, 41, 71, 20];
- sistemas de potencia [22, 23, 15];
- telecomunicaciones [2, 13, 64, 70, 88, 89, 94, 26, 82, 104, 65, 72];
- diseño de redes [8], [34];
- dibujo de grafos y mapas [47, 67, 93, 96, 73, 74, 24];
- lenguaje [31];
- estadística [79, 80];
- biología [9];
- programación matemática [83];
- empaquetado [30]; y
- VLSI [10], entre otras áreas de aplicación.

7 Bibliografía

[1] S. Abdinnour-Helm and S.W. Hadley. Tabu search based heuristics for multi-floor facility layout. International Journal of Production Research, 38:365-383, 2000.

[2] J. Abello, P.M. Pardalos, and M.G.C. Resende. On maximum clique problems in very large graphs. In J. Abello and J. Vitter, editors, External Memory Algorithms and Visualization, volume 50 of DIMACS Series on Discrete Mathematics and Theoretical Computer Science, pages 199-130. American Mathematical Society, 1999.

[3] R.K. Ahuja, J.B. Orlin, and D. Sharma. New neighborhood search structures for the capacitated minimum spanning tree problem. Mathematical Programming, 91:71-97, 2001.

[4] R.M. Aiex, S. Binato, and M.G.C. Resende. Parallel GRASP with path relinking for job shop scheduling. Parallel Computing, 29:393- 430, 2003.

[5] R.M. Aiex, M.G.C. Resende, P.M. Pardalos, and G. Toraldo. GRASP with path relinking for the three-index assignment problem. Technical report, AT&T Labs-Research, 2000. To appear in INFORMS J. on Comp.

[6] M.S. Akturk and D. Ozdemir. A new dominance rule to minimize total weighted tardiness with unequal release dates. European Journal of Operational Research, 135:394-412, 2001.

[7] lvarez, A., González Velarde, J.L., de Alba, K. GRASP Embedded Scatter Search for the Multicommodity Capacitated Network Design Problem. Journal of Heuristics, 11:233-257, 2005.

[8] A. Andreatta and C.C. Ribeiro. Heuristics for the phylogeny problem. Journal of Heuristics, 8:429-447, 2002.

[9] S. Areibi and A. Vannelli. A GRASP clustering technique for circuit partitioning. In J. Gu and P.M. Pardalos, editors, Satisfiability Problems, volume 35 of DIMACS Series on Discrete Mathematics and Theoretical Computer Science, pages 711-724. American Mathematical Society, 1997.

[10] M.F. Argüello, J.F. Bard, and G. Yu. A GRASP for aircraft routing in response to groundings and delays. Journal of Combinatorial Optimization, 1:211-228, 1997.

[11] M.F. Argüello, T.A. Feo, and O. Goldschmidt. Randomized methods for the number partitioning problem. Computers and Operations Research, 23:103-111, 1996.

[12] M. Armony, J.C. Klincewicz, H. Luss, and M.B. Rosenwein. Design of stacked selfhealing rings using a genetic algorithm. Journal of Heuristics, 6:85-105, 2000.

[13] J.B. Atkinson. A greedy randomised search heuristic for time- constrained vehicle scheduling and the incorporation of a learning strategy. Journal of the Operational Research Society, 49:700-708, 1998.

[14] L. Bahiense, G.C. Oliveira, and M. Pereira. A mixed integer disjunctive model for transmission network expansion. IEEE Transactions on Power Systems, 16:560-565, 2001.

[15] J.F. Bard and T.A. Feo. Operations sequencing in discrete parts manufacturing. Management Science, 35:249-255, 1989.

[16] J.F. Bard and T.A. Feo. An algorithm for the manufacturing equipment selection problem. IIE Transactions, 23:83-92,1991.

[17] J.F. Bard, T.A. Feo, and S. Holland. A GRASP for scheduling printed wiring board assembly. IIE Transactions, 28:155-165, 1996.

[18] J.F. Bard, L. Huang, P. Jaillet, and M. Dror. A decomposition approach to the inventory routing problem with satellite facilities. Transportation Science, 32:189–203, 1998.

[19] J.F. Bard, G. Kantoravdis, and G. Yu. A branch-and-cut procedure for the vehicle routing problem with time windows. Transportation Science, 36:250-269, 2002.

[20] S. Binato, W.J. Hery, D. Loewenstern, and M.G.C. Resende. A GRASP for job shop scheduling. In C.C. Ribeiro and P. Hansen, editors, Essays and Surveys in Metaheuristics, pages 59-79. Kluwer Academic Publishers, 2002.

[21] S. Binato and G.C. Oliveira. A reactive GRASP for transmission network expansion planning. In C.C. Ribeiro and P. Hansen, editors, Essays and Surveys in Metaheuristics, pages 81-100. Kluwer Academic Publishers, 2002.

[22] S. Binato, G.C. Oliveira, and J.L. Araújo. A greedy randomized adaptive search procedure for transmission expansion planning. IEEE Transactions on Power Systems, 16:247-253, 2001.

[23] C. Binucci, W. Didimo, G. Liotta, and M. Ñonato. Labeling heuristics for orthogonal drawings. In Proceedings of GD'98 - Symposium on Graph Drawing, volume 2265 of Lecture Notes in Computer Science, pages 139-153. Springer-Verlag, 2002.

[24] J.L. Bresina. Heuristic-biased stochastic sampling. In Proceedings of the Thirteenth National Conference on Artificial Intelligence, pages 271-278, Portland, 1996.

[25] M. Brunato and R. Battiti. A multistart randomized greedy algorithm for traffic grooming on mesh logical topologies. Technical report, Department of Mathematics, University of Trento, Trento, Italy, 2001.

[26] S.A. Canuto, M.G.C. Resende, and C.C. Ribeiro. Local search with perturbations for the prize-collecting Steiner tree problem in graphs. Networks, 38:50-58, 2001.

[27] S.A. Canuto, C.C. Ribeiro, and M.G.C. Resende. Local search with perturbations for the prize-collecting Steiner tree problem. In Extended Abstracts of the Third Metaheuristics International Conference, pages 115-119, Angra dos Reis, July 1999.

[28] C. Carreto and B. Baker. A GRASP interactive approach to the vehicle routing problem with backhauls. In C.C. Ribeiro and P. Hansen, editors, Essays and Surveys in Metaheuristics, pages 185- 199. Kluwer Academic Publishers, 2002.

[29] P. Chardaire, G.P. McKeown, and J.A. Maki. Application of GRASP to the multiconstraint knapsack problem. In E.J.W. Boers et al., editor, Evo-Workshop 2001, pages 30–39. Springer-Verlag Berlin Heidelberg, 2001.

[30] H. Fraçois and O. Boëffard. The greedy algorithm and its application to the construction of a continuous speech database. In Proceedings of LREC-2002, volume 5, pages 1420-1426, May 29-31 2002.

[31] A. Corberán, R. Martí, and J.M. Sanchís. A GRASP heuristic for the mixed Chinese postman problem. European Journal of Operational Research, 142:70-80, 2002.

[32] P. De, J.B. Ghosj, and C.E. Wells. Solving a generalized model for con due date assignment and sequencing. International Journal of Production Economics, 34:179-185, 1994.

[33] De Alba, K. lvarez, A., González Velarde, J.L. GRASP with Adaptive Memory for finding good starting solutions for the capacitated network design problem, en el libro Computational Modeling and Problem Solving in the Networked World:, pp 121 - 137. Series Interfaces in Computing and Optimization Editors: Hemant K. Bhargava and Nong Ye, Kluwer 2003

[34] H. Delmaire, J.A. Díaz, E. Fernández, and M. Ortega. Reactive GRASP and Tabu Search based heuristics for the single source capacitated plant location problem. INFOR, 37:194-225, 1999.

[35] A.S. Deshpande and E. Triantaphyllou. A greedy randomized adaptive search procedure (GRASP) for inferring logical clauses from examples in polynomial time and some extensions. Mathematical Computer Modelling, 27:75-99, 1998

[36] A. Drexl and F. Salewski. Distribution requirements and compactness constraints in school timetabling. European Journal of Operational Research, 102:193-214, 1997.

[37] T.A. Feo and J.F. Bard. Flight scheduling and maintenance base planning. Management Science, 35:1415-1432, 1989.

[38] T.A. Feo and J.F. Bard. The cutting path and tool selection problem in computer aided process planning. Journal of Manufacturing Systems, 8:17-26, 1989.

[39] T.A. Feo, J.F. Bard, and S. Holland. Facility-wide planning and scheduling of printed wiring board assembly. Operations Research, 43:219-230, 1995.

[40] T.A. Feo and J.L. González-Velarde. The intermodal trailer assignment problem: Models, algorithms, and heuristics. Transportation Science, 29:330-341,1995.

[41] T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally dificult set covering problem. Operations Research Letters, 8:67–71, 1989.

[42] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. Journal of Global Optimization, 6:109-133, 1995

[43] T.A. Feo, M.G.C. Resende, and S.H. Smith. A greedy randomized adaptive search procedure for maximum independent set. Operations Research, 42:860-878, 1994.

[44] T.A. Feo, K. Sarathy, and J. McGahan. A GRASP for single machine scheduling with sequence dependent setup costs and linear delay penalties. Computers and Operations Research, 23:881-895, 1996.

[45] T.A. Feo, K. Venkatraman, and J.F. Bard. A GRASP for a difficult single machine scheduling problem. Computers and Operations Research, 18:635-643, 1991.

[46] E. Fernández and R. Martí. GRASP for seam drawing in mosaicking of aerial photographic maps. Journal of Heuristics, 5:181-197, 1999.

[47] P. Festa, P.M. Pardalos, and M.G.C. Resende. Algorithm 815: FORTRAN subroutines for computing approximate solution to feedback set problems using GRASP. ACM Transactions on Mathematical Software, 27:456-464, 2001.

[48] P. Festa, P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro. Randomized heuristics for the MAX-CUT problem. Optimization Methods & Software, 7:1033-1058, 2002.

[49] P. Festa and M.G.C. Resende. GRASP: An annotated bibliography. In C.C. Ribeiro and P. Hansen, editors, Essays and Surveys in Metaheuristics, pages 325-367. Kluwer Academic Publishers, 2002.

[50] C. Fleurent and F. Glover. Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. INFORMS Journal on Computing, 11:198-204, 1999.

[51] J.B. Ghosh. Computational aspects of the maximum diversity problem. Operations Research Letters, 19:175-181, 1996.

[52] F. Glover. Tabu search and adaptive memory programing : Advances, applications and challenges. In R.S. Barr, R.V. Helgason, and J.L. Kennington, editors Interfaces in Computer Science and Operations Research, pages 1-75. Kluwer, 1996.

[53] P.L. Hammer and D.J. Rader, Jr. Maximally disjoint solutions of the set covering problem. Journal of Heuristics, 7:131-144, 2001.

[54] J.P. Hart and A.W. Shogan. Semi-greedy heuristics: An empirical study. Operations Research Letters, 6:107-114, 1987.

[55] M. Hasan, I. Osman, and T. AlKhamis. A meta-heuristic procedure for the three dimension assignment problem. International Journal of Applied Mathematics, 8:365-380, 2002.

[56] Hirsch, M.J., Meneses, C.N., Pardalos, P.M., and Resende, M.G.C., Global Optimization by Continuous Grasp, to appear in Optimization Letters, 2006.

[57] M. J. Hirsch, P. M. Pardalos, and M. G. C. Resende, Solving systems of nonlinear equations using Continuous GRASP. Submitted to Journal of Heuristics, 2006.

[58] M. J. Hirsch, P. M. Pardalos, and M. G. C. Resende. Speeding up Continuous GRASP. Submitted to European Journal of Operational Research, 2006.

[59] M. J. Hirsch, C. N. Meneses, P. M. Pardalos, M. A. Ragle, M. G. C. Resende A continuous GRASP to determine the relationship between drugs and adverse reactions. Optimization on Line, http://www.optimization-online.org/DB_HTML/2006/10/1495.html, 2006

[60] J.G. Klincewicz. Avoiding local optima in the p-hub location problem using tabu search and GRASP. Annals of Operations Research, 40:283-302, 1992.

[61] J.G. Klincewicz. Enumeration and search procedures for a hub location problem with economies of scale. Annals of Operations Research, 110:107-122, 2002.

[62] J.G. Klincewicz and A. Rajan. Using GRASP to solve the component grouping problem. Naval Research Logistics, 41:893-912, 1994.

[63] G. Kontoravdis and J.F. Bard. A GRASP for the vehicle routing problem with time windows. ORSA Journal on Computing, 7:10-23, 1995.

[64] M. Laguna, T.A. Feo, and H.C. Elrod. A greedy randomized adaptive search procedure for the two-partition problem. Operations Research, 42:677-687, 1994.

[65] M. Laguna and J.L. González Velarde. A search heuristic for just-in-time scheduling in parallel machines. Journal of Intelligent Manufacturing, 2:253-260, 1991.

[66] M. Laguna and R. Martí. GRASP and path relinking for 2- layer straight line crossing minimization. INFORMS Journal on Computing, 11:44-52, 1999.

[67] M. Laguna and R. Martí. A GRASP for coloring sparse graphs. Computational Optimization and Applications, 19:165-178, 2001.

[68] Y. Li, P.M. Pardalos, and M.G.C. Resende. A greedy randomized adaptive search procedure for the quadratic assignment problem. In P.M. Pardalos and H. Wolkowicz, editors, Quadratic Assignment and Related Problems, volume 16 of DIMACS Series on Discrete Mathematics and Theoretical Computer Science, pages 237-261. American Mathematical Society, 1994.

[69] X. Liu, P.M. Pardalos, S. Rajasekaran, and M.G.C. Resende. A GRASP for frequency assignment in mobile radio networks. In B.R. Badrinath, F. Hsu, P.M. Pardalos, and S. Rajasejaran, editors, Mobile Networks and Computing, volume 52 of DIMACS Series on Discrete Mathematics and Theoretical Computer Science, pages 195- 201. American Mathematical Society, 2000.

[70] H. Ramalhinho Lourenço, J.P. Paixão, and R. Portugal. Multiobjective metaheuristics for the bus-driver scheduling problem. Transportation Sciences, 35:331-343, 2001.

[71] P. Mahey and C.C. Ribeiro. Optimal routing for multi- service communication networks. OR/MS Today, 29:40-43, 2002.

[72] R. Martí. Arc crossing minimization in graphs with GRASP. IIE Transactions, 33:913-919, 2001.

[73] R. Martí and V. Estruch. Incremental bipartite drawing problem. Computers and Operations Research, 28:1287-1298, 2001.

[74] S.L. Martins, P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro. Greedy randomized adaptive search procedures for the Steiner problem in graphs. In P.M. Pardalos, S. Rajasejaran, and J. Rolim, editors, Randomization Methods in Algorithmic Design, volume 43 of DIMACS Series on Discrete Mathematics and Theoretical Computer Science, pages 133-145. American Mathematical Society, 1999.

[75] S.L. Martins, M.G.C. Resende, C.C. Ribeiro, and P. Pardalos. A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy. Journal of Global Optimization, 17:267-283, 2000.

[76] S.L. Martins, C.C. Ribeiro, and M.C. Souza. A parallel GRASP for the Steiner problem in graphs. In A. Ferreira and J. Rolim, editors, Proceedings of IRREGULAR'98 - 5th International Symposium on Solving Irregularly Structured Problems in Parallel, volume 1457 of Lecture Notes in Computer Science, pages 285-297. Springer- Verlag, 1998.

[77] T. Mavridou, P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A GRASP for the biquadratic assignment problem. European Journal of Operational Research, 105:613-621, 1998.

[78] M.C. Medeiros, M.G.C. Resende, and A. Veiga. Piecewise linear time series estimation with GRASP. Computational Optimization and Applications, 19:127-144, 2001.

[79] M.C. Medeiros, A. Veiga, and M.G.C. Resende. A combinatorial approach to piecewise linear time analysis. Journal of Computational and Graphical Statistics, 11:236-258, 2002.

[80] R.A. Murphey, P.M. Pardalos, and L.S. Pitsoulis. A parallel GRASP for the data association multidimensional assignment problem. In P.M. Pardalos, editor, Parallel Processing of Discrete Problems, volume 106 of The IMA Volumes in Mathematics and Its Applications, pages 159-180. Springer-Verlag, 1998.

[81] A. Myslek. Greedy randomised adaptive search procedures (GRASP) for topological design of mpls networks. In Proceedings of the 8th Polish Teletraffic Symposium, 2001.

[82] G. Palubeckis and A. Tomkevicius. GRASP implementations for the unconstrained binary quadratic optimization problem. Information Technology and Control, 24:14-20, 2002.

[83] P. M. Pardalos, T. Qian, and M. G. C. Resende. A greedy randomized adaptive search procedure for the feedback vertex set problem. Journal of Combinatorial Optimization, 2:399-412, 1999.

[84] P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A parallel GRASP implementation for the quadratic assignment problem. In A. Ferreira and J. Rolim, editors, Parallel Algorithms for Irregularly Structured Problems - Irregular'94, pages 115-133. Kluwer Academic Publishers, 1995.

[85] P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A parallel GRASP for MAXSAT problems. Lecture Notes in Computer Science, 1184:575-585, 1996.

[86] L.S. Pitsoulis, P.M. Pardalos, and D.W. Hearn. Approximate solutions to the turbine balancing problem. European Journal of Operational Research, 130:147-155, 2001.

[87] M. Prais and C.C. Ribeiro. Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. INFORMS Journal on Computing, 12:164-176, 2000.

[88] M.G.C. Resende. Computing approximate solutions of the maximum covering problem using GRASP. Journal of Heuristics, 4:161-171, 1998.

[89] M.G.C. Resende and T.A. Feo. A GRASP for satisfiability. In D.S. Johnson and M.A. Trick, editors, Cliques, Coloring, and Satisfiability: The Second DIMACS Implementation Challenge, volume 26 of DIMACS Series on Discrete Mathematics and Theoretical Computer Science, pages 499-520. American Mathematical Society, 1996.

[90] M.G.C. Resende, L.S. Pitsoulis, and P.M. Pardalos. Approximate solution of weighted MAX-SAT problems using GRASP. In J. Gu and P.M. Pardalos, editors, Satisfiability Problems, volume 35 of DIMACS Series on Discrete Mathematics and Theoretical Computer Science, pages 393-405. American Mathematical Society, 1997.

[91] M.G.C. Resende, L.S. Pitsoulis, and P.M. Pardalos. Fortran subroutines for computing approximate solutions of MAXSAT problems using GRASP. Discrete Applied Mathematics, 100:95-113, 2000.

[92] M.G.C. Resende and C.C. Ribeiro. A GRASP for graph planarization. Networks, 29:173-189, 1997.

[93] M.G.C. Resende and C.C. Ribeiro. A GRASP with path- relinking for private virtual circuit routing. Networks, 41(1):104-114, 2003.

[94] M.G.C. Resende and R.F. Werneck. A GRASP with path- relinking for the p-median problem. Technical report, Internet and Network Systems Research Center, AT&T Labs Research, Florham Park, NJ, 2002.

[95] C.C. Ribeiro and M.G.C. Resende. Algorithm 797: Fortran subroutines for approximate solution of graph planarization problems using GRASP. ACM Transactions on Mathematical Software, 25:342-352, 1999.

[96] C.C. Ribeiro, E. Uchoa, and R.F. Werneck. A hybrid GRASP with perturbations for the Steiner problem in graphs. INFORMS Journal on Computing, 14:228-246, 2002.

[97] R.Z. Ríos Mercado and J.F. Bard. Heuristics for the flow line problem with setup costs. European Journal of Operational Research, pages 76-98, 1998.

[98] R.Z. Ríos Mercado and J.F. Bard. An enhanced TSP-based heuristic for makespan minimization in a flow shop with setup costs. Journal of Heuristics, 5:57-74, 1999.

[99] A.J. Robertson. A set of greedy randomized adaptive local search procedure (GRASP) implementations for the multidimensional assignment problem. Computational Optimization and Applications, 19:145-164, 2001.

[100] D. Serra and R. Colomé. Consumer choice in competitive location models: Formulations and heuristics. Papers in Regional Science, 80:439-464, 2001.

[101] T.L. Urban. Solution procedures for the dynamic facility layout problem. Annals of Operations Research, pages 323-342, 1998.

[102] T.L. Urban, W.-C. Chiang, and R.A. Russel. The integrated machine allocation and layout problem. International Journal of Production Research, pages 2913-2930, 2000.

[103] J. Xu and S. Chiu. Effective heuristic procedure for a field technician scheduling problem. Journal of Heuristics, 7:495-509, 2001.

[104] J. Yen, M. Carlsson, M. Chang, J.M. Garcia, and H. Nguyen. Constraint solving for inkjet print mask design. Journal of Imaging Science and Technology, 44:391-397, 2000.

Principios de la Búsqueda Dispersa*

S. Casado[a] **y R. Martí**[b]

[a]Dpto. de Economía Aplicada,
Universidad de Burgos,
Plaza Infanta Elena s/n 09001 Burgos,
scasado@ubu.es ,
[b]Dpto. de Estadística e Investigación Operativa,
Facultad de Matemáticas,
Universidad de Valencia,
Dr. Moliner 50, 46100 Burjassot (Valencia),
Rafael.Marti@uv.es Estadística e Investigación Operativa

1 Introducción

La Búsqueda Dispersa es un método que fue introducido en 1977 (publicado en Glover 1998) como un heurístico para la programación entera, basado en estrategias expuestas en el congreso "Management Science and Engineering Management" celebrado en Austin, Texas en septiembre de 1967. A pesar de eso SS no fue aplicado ni debatido hasta 1990, cuando fue presentado en el "EPFL Seminar on Operations Research and Artificial Intelligence Search Methods" (Lausanne, Switzerland). Un artículo basado en esta exposición fue publicado en 1994 (Glover 1994), y desde entonces la metodología de Scatter Search se empezó a aplicar con más profusión.

La primera descripción de SS (Glover, 1977) usa una sucesión de principios coordinados para generar soluciones. Concretamente los aspectos más destacados

*Los autores de este trabajo agradecen la ayuda del Ministerio de Educación y Ciencia por la subvención económica para la realización de este trabajo a través del Plan Nacional de I+D, (Proyecto SEJ- 2005 08923/ECON), así como la recibida de la Conserjería de Educación de Castilla y León (Proyecto BU008A06).

de este trabajo son los siguientes:

- SS realiza una exploración sistemática sobre una serie de buenas soluciones llamadas conjunto de referencia teniendo en cuenta las características de diversos elementos de la solución .
- El método se centra en combinar dos o más soluciones del conjunto de referencia. La combinación de más de dos soluciones tiene como objetivo el generar centroides.
- Generar soluciones en la línea que une dos dadas se considera una forma reducida del método.
- Al combinar se deben seleccionar pesos apropiados y no tomar valores al azar.
- Se deben realizar combinaciones "convexas" y "no convexas" de las soluciones.
- La distribución de los puntos se considera importante y deben tomarse dispersos.

Básicamente se trata de que mediante la combinación de las soluciones que forman el conjunto de referencia se obtengan nuevas soluciones que mejoren a las que las originaron. Según esto, cuando por ejemplo se crean nuevas soluciones a partir de una combinación lineal de otras dos o más, el conjunto de referencia puede evolucionar según se observa en la Figura 1.

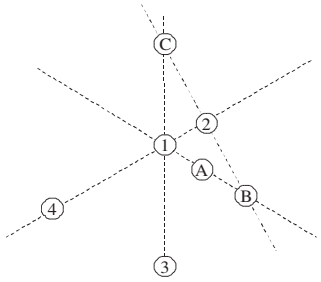


Fig.1.- Conjunto de Referencia

Considerando el conjunto de referencia original como el formado por las soluciones etiquetadas como A, B y C, una combinación no convexa de las soluciones de referencia A y B crea la solución 1. En realidad se crean más soluciones en el segmento definido por A y B, aunque sólo se introduce en el conjunto de referencia la solución 1 (el criterio usado para seleccionar las soluciones que forman parte del conjunto de referencia será tratado más adelante). De igual forma, mediante las combinaciones convexas y no convexas de las soluciones del conjunto de referencia y la recién creada se originan los puntos 2, 3 y 4. El conjunto de referencia

completo mostrado en la Figura 1 consta de 7 soluciones que posteriormente serán mejoradas con un procedimiento de búsqueda local.

Históricamente los antecedentes de las estrategias para combinar soluciones fueron introducidos en el contexto de los métodos de planificación para obtener mejoras en las soluciones para *"job shop scheduling problems"* (Glover 1963). Se generaron nuevas soluciones mediante la creación de combinaciones numéricamente ponderadas de las soluciones ya existentes. Esta técnica fue motivada por la suposición de que la información está contenida de diferentes formas en diferentes soluciones y por ello esta información puede ser utilizada de un modo más efectivo cuando se combina que cuando la tratamos con estrategias estándar de selección de diferentes soluciones sin tener en cuenta el resto.

Además, en programación entera y no lineal, se desarrollaron procedimientos asociados para combinar restricciones. Hoy en día son muy conocidos los métodos para crear nuevas restricciones de desigualdad llamados *surrogate constraints* (restricciones subrogadas) (Glover 1965 y 1968).

Otros mecanismos de combinación son aquellos basados en "votos". En éstos se definen reglas mediante las que cada solución "vota" para que sus características aparezcan en la solución que se está construyendo. Estos métodos de votos han sido muy utilizados en las rutinas de combinación de SS y parece que constituyen una de las claves de su éxito.

Esta forma de operar de SS combinando soluciones hace que se le enmarque dentro de los métodos que llamamos "evolutivos" o "basados en población", es decir, aquellos que combinan soluciones para crear otras nuevas. De hecho se basa en el principio de que la información sobre la calidad o el atractivo de un conjunto de soluciones puede ser utilizado mediante la combinación de éstas. En concreto, dadas dos soluciones, se puede obtener una nueva mediante su combinación de modo que mejore a las que la originaron.

A pesar de que SS sea un método de los que denominamos "evolutivos" existen algunas diferencias importantes entre los algoritmos genéticos, probablemente el representante más conocido y extendido de los algoritmos evolutivos, y SS. Básicamente son las siguientes:

- Mientras que en SS la selección de las soluciones se hace de forma sistemática y estratégica, en los algoritmos genéticos se realiza de forma totalmente aleatoria.
- SS selecciona las soluciones, para combinarlas posteriormente, de entre un pequeño conjunto de soluciones denominado conjunto de referencia, mientras que los algoritmos genéticos consideran una población de soluciones de mayor tamaño. Así, los algoritmos genéticos suelen considerar una población de 100 soluciones mientras que en SS habitualmente se trabaja con conjuntos de referencia de 10 soluciones.
- Podemos identificar aquí otra diferencia con el resto de métodos evolutivos

en los que con frecuencia se emplean métodos de combinación independientes del contexto. Es decir que no utilizan ninguna información o conocimiento sobre el problema, como el conocido operador de sobrecruzamiento (crossover) en los algoritmos genéticos.

Por otro lado, el método SS no sólo consiste en combinar soluciones del conjunto de referencia sino que va más allá y a las soluciones obtenidas tras la combinación se les aplica un procedimiento de mejora que habitualmente es un procedimiento de búsqueda local, aunque en diseños avanzados se puede incorporar al procedimiento de mejora estructuras de memoria. Esta forma de actuar está basada en la suposición de que cuando se combinan soluciones y se aplica un método de mejora sobre las mismas se obtienen mejores resultados que cuando se aplica el método de mejora en las soluciones originales sin haberlas combinado previamente.

En Glover (1998) se recopilan y organizan ideas fundamentales de SS procedentes de trabajos anteriores dando lugar a una versión estándar del método mediante un esquema o plantilla. De ahí la gran importancia de la publicación de este trabajo en lo que se refiere a la difusión del método. Básicamente destacamos las siguientes ideas:

- La información útil sobre la forma o la situación de las soluciones óptimas está normalmente contenida en un conjunto apropiado y diverso de soluciones élite.
- Cuando la combinación de soluciones se usa como una estrategia para explotar tal información, es importante incorporar mecanismos capaces de generar combinaciones que vayan más allá de las regiones abarcadas por las soluciones consideradas. De un modo similar también es importante incorporar procesos heurísticos para transformar las soluciones combinadas en nuevas soluciones. El objetivo de estos mecanismos de combinación es incorporar diversidad y calidad.
- Si se tienen en cuenta múltiples soluciones simultáneamente como base para crear combinaciones, se intensifica la oportunidad de explotar información contenida en la unión de soluciones élite.

2 Método Básico

Scatter Search trabaja sobre un conjunto pequeño de soluciones, denominado conjunto de referencia, combinando sus soluciones para crear otras nuevas. A continuación se describen los cinco elementos esenciales del método así como el funcionamiento de estos elementos dentro del esquema básico de SS.

1. Método generador de soluciones diversas. Con este método se genera un conjunto de soluciones diversas que en principio no tienen que ser necesaria-

mente factibles. El tamaño de este conjunto, P, suele estar en torno a 100 aunque depende de variantes.

2. Conjunto de referencia (Refset). De entre el conjunto de soluciones diversas generado con el método anterior y una vez aplicado el método de mejora, se selecciona el conjunto de referencia, formado por un número pequeño de soluciones, b (alrededor de $b = 10$). La mitad de éstas soluciones ($b/2$) serán las de mayor calidad del conjunto de soluciones diversas y la otra mitad se obtiene siguiendo el criterio de la diversidad, es decir se seleccionan aquellas que disten más (según la medida de diversidad considerada en el problema) respecto a las ya incluidas en el conjunto de referencia. Las soluciones seleccionadas se ordenan según su calidad de mayor a menor.

3. Un método generador de subconjuntos. A través de este método se generan subconjuntos de soluciones del conjunto de referencia. Las soluciones de cada uno de estos subconjuntos se combinarán entre sí posteriormente. Un criterio seguido en numerosas ocasiones para obtener los subconjuntos consiste en considerar todos los pares de soluciones del conjunto de referencia, aunque se pueden considerar tríos o subconjuntos formados por cualquier otro número de soluciones.

4. Un método de combinación. Con este método se combinan entre sí las soluciones de cada subconjunto obtenido con el método generador de subconjuntos ya descrito.

5. Método de mejora de soluciones. Este método se usa para tratar de obtener soluciones de mayor calidad que las de partida, aunque en el caso en que aparezcan soluciones no factibles su función consistirá, primero en obtener una solución factible y luego intentar mejorarla. Se aplica tanto al conjunto de soluciones diversas como a aquellas soluciones que se obtienen tras la aplicación del método de combinación a las del conjunto de referencia. Habitualmente como método de mejora se usa un procedimiento de búsqueda local.

El modo de actuación de los elementos descritos anteriormente dentro del esquema básico del algoritmo de SS se muestra a continuación.

Algoritmo de Scatter Search

1. **Generación** de un conjunto de P soluciones diversas
2. **Mejora** de dichas soluciones
3. Construcción del **conjunto de referencia** con las b mejores soluciones siguiendo los criterios de calidad y diversidad.
4. Repetir
   - 4.1 **Formación de subconjuntos** con las soluciones del conjunto de referencia.
   - 4.2 Generación de soluciones nuevas mediante la aplicación del **método de combinación** a las soluciones de los subconjuntos (para obtener soluciones distintas a las de partida)
   - 4.3 **Mejora** de las nuevas soluciones
   - 4.4 **Actualización** del conjunto de referencia. (Las nuevas soluciones obtenidas que sean buenas por calidad o por diversidad se incorporan al conjunto de referencia)

   Hasta que el conjunto de referencia se estabilice (esto ocurre si durante un ciclo completo no se obtiene ninguna solución que pueda ser incorporada en el mismo)

Tal y como se observa en el esquema del algoritmo de SS, de la combinación de las soluciones del conjunto de referencia se obtienen nuevas soluciones que una vez mejoradas pueden pasar a formar parte del conjunto de referencia, actualizando así dicho conjunto. Dado que el número de soluciones del conjunto de referencia no varía a lo largo de todo procedimiento, la actualización de este conjunto se realizará de forma que las nuevas soluciones sustituirán a las que mejoren en el conjunto de referencia. Es importante destacar que el significado de *"mejores"* no se restringe a la calidad de la solución, sino que también se considera la diversidad que ésta aporta al conjunto. No obstante, aunque la actualización se puede hacer según el criterio de diversidad se ha comprobado que siguiendo el criterio de calidad se obtienen mejores resultados (Laguna y Martí 2003). De esta forma la calidad de las soluciones del conjunto de referencia puede ir mejorando progresivamente.

Finalmente el algoritmo se detiene cuando el conjunto de referencia se estabiliza, es decir, cuando durante un ciclo completo no se obtiene ninguna solución que pueda pasar a formar parte del conjunto de referencia. Llegado a este punto

el algoritmo puede reiniciarse volviendo al paso 1 del esquema y repitiendo todo el procedimiento. Una práctica habitual para reiniciar el proceso consiste en obtener un nuevo conjunto de referencia de la manera siguiente: Partiendo del conjunto de referencia que se había estabilizado (no admitía ninguna nueva solución) se eliminan la mitad de las soluciones ($b/2$), manteniéndose la otra mitad. Concretamente se eliminan las de peor calidad. Para obtener la otra mitad de las soluciones se genera un nuevo conjunto de P soluciones (paso 1 del algoritmo) considerando como objetivo favorecer la diversidad respecto a las soluciones que no se han eliminado. De este conjunto, en sucesivos pasos, se selecciona la solución más diversa respecto a las que ya forman parte del nuevo conjunto de referencia hasta que éste llegue a tener b soluciones. Una vez obtenido el nuevo conjunto de referencia se continúa con el paso 4 (4.1, 4.2, 4.3 y 4.4) siguiendo el esquema del algoritmo. En el gráfico de la Figura 2 se muestra el funcionamiento básico del algoritmo de SS.

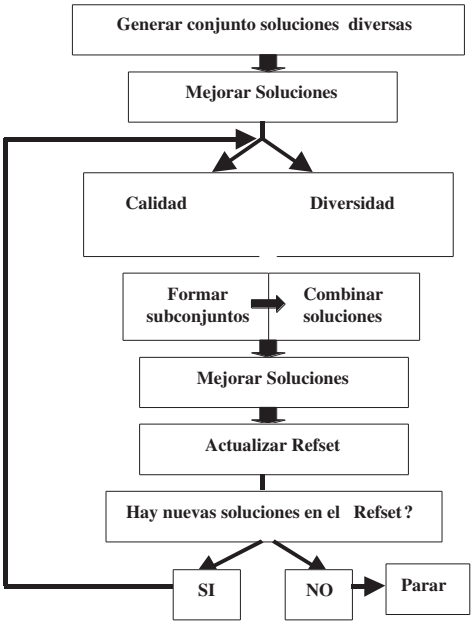


Fig.2.- Funcionamiento del algoritmo de SS

3 Estrategias Avanzadas

Scatter Search puede ser implementado de múltiples formas y ofrece alternativas muy diversas para explotar sus ideas fundamentales. De hecho su mecanismo no está restringido a un diseño único y uniforme, sino que permite diferentes posibilidades que pueden resultar efectivas según el caso.

SS proporciona, por tanto, un marco flexible que permite el desarrollo de diferentes variantes con diversos grados de complejidad entre las que se pueden destacar las siguientes propuestas:

1) Dependiendo del momento en que se realice la actualización del conjunto de referencia podemos distinguir dos variantes:

a) Actualización Estática: La actualización del conjunto de referencia se realiza una vez que se han combinado todos los subconjuntos y se han obtenido todas las nuevas soluciones. De este modo hasta ese momento no se podrá saber si una solución determinada se va a poder incorporar al conjunto de referencia. Básicamente su funcionamiento se resume en el siguiente esquema:

Algoritmo Scatter Search "estático"

1. Generar un conjunto inicial P de soluciones diversas
2. Mejorar las soluciones generadas
3. Con estas soluciones construir el conjunto de referencia inicial
4. Repetir
   - 4.1 Obtener todos los subconjuntos de elementos del conjunto de referencia
   - 4.2 Combinar las soluciones de cada subconjunto para obtener nuevas soluciones
   - 4.3 Mejorar las nuevas soluciones obtenidas
   - 4.4 Actualizar el conjunto de referencia con estas nuevas soluciones hasta que se estabilice (i.e. no se incluyan nuevas soluciones)

---

b) Actualización Dinámica: La actualización del conjunto de referencia se realiza cada vez que se genera una nueva solución, de modo que en el momento en que obtiene una solución se sabe si ésta va a formar parte del conjunto de referencia. Esta segunda variante es más agresiva dado que cada vez que se genera una solución apta para entrar en el conjunto de referencia ésta pasa inmediatamente a formar parte de él, en vez de esperar a que se hayan combinado todos los subconjuntos de soluciones. La ventaja de este tipo de actualización es que en el caso en que el conjunto de referencia contenga soluciones de baja calidad, esas soluciones pueden ser reemplazadas rápidamente y así en las siguientes combinaciones intervendrán soluciones de mayor calidad. Sin embargo, el hecho de que el tamaño de este conjunto se deba mantener constante implica que puedan existir soluciones que salgan del conjunto de referencia sin haber sido combinadas.

Una vez descritas ambas variantes, observamos cómo implementar la variante dinámica resulta ser más complejo que la variante estática. Además, en la variante estática el orden en el cual tiene lugar la formación de subconjuntos y combinación de soluciones carece de relevancia dado que se generan todos los subconjuntos y se realizan todas las combinaciones. Sin embargo, en la actualización dinámica este orden tiene una gran importancia dado que determina la eliminación de combinaciones potenciales. Por esto, cuando se actualiza según la variante dinámica, es necesario hacer pruebas siguiendo distintos órdenes de formación de subconjuntos y combinación de soluciones.

A continuación se describe el esquema básico de funcionamiento de la variante "dinámica" de SS :

Algoritmo Scatter Search "dinámico"

1. Generación de un conjunto P de soluciones diversas
2. Mejorar las soluciones generadas
3. Construir el conjunto de referencia inicial Refset
4. Repetir
   - 4.1 Siguiendo el orden establecido de formación de subconjuntos, hacer:
     - 4.1.1 Obtener el subconjunto de elementos que corresponda
     - 4.1.2 Combinar las soluciones de este subconjunto para obtener una nueva solución
     - 4.1.3 Mejorar esta nueva solución
     - 4.1.4 Actualizar el conjunto de referencia: la solución mejorada se incorpora al Refset si es mejor que la peor en él. hasta que se estabilice el conjunto de referencia (i.e. ninguna nueva solución pueda ser incluida en el Refset)

---

2) Se pueden implementar diferentes diseños en función del número de elementos que integran los subconjuntos de soluciones que se obtienen del conjunto de referencia. Lo más común y que permite obtener los mejores resultados es que estén formados por dos elementos (se suelen considerar todos los pares de soluciones posibles). De hecho, en el experimento presentado en Campos y otros (2001) se muestra como al menos el 80% de las soluciones que son admitidas en el conjunto de referencia provienen de combinaciones de subconjuntos formados por dos elementos. No obstante los subconjuntos pueden estar formados por cualquier otro número de elementos o soluciones (por ejemplo subconjuntos de 3 elementos, de cuatro, etc.)

Un procedimiento práctico para generar subconjuntos que permite pasar de subconjuntos de dos elementos a otros formados por un número mayor, teniendo siempre controlado el número de subconjuntos que se van a generar, consiste en crearlos de la siguiente forma propuesta en Glover (1998):

- Subconjuntos de dos elementos: formados por todos los pares de soluciones posibles.
- Subconjuntos de tres elementos: derivados de los subconjuntos de dos elementos añadiendo a cada uno de ellos la mejor solución encontrada que no pertenezca al subconjunto.
- Subconjuntos de cuatro elementos: derivados de los subconjuntos de tres elementos añadiendo a cada uno de ellos la mejor solución encontrada que no pertenezca al subconjunto .
- Subconjuntos formados por los i mejores elementos, desde $i = 5$ hasta b.

3) Según la forma de actualizar el conjunto de referencia: Este conjunto inicialmente se forma siguiendo los criterios de calidad y diversidad, de modo que generalmente aproximadamente la mitad de los elementos lo constituyan las soluciones de mayor calidad y el resto se obtengan según el criterio de máxima distancia. Sin embargo a la hora de actualizarlo lo más habitual es hacerlo sólo siguiendo el criterio de calidad. De hecho tal y como se ha comentado anteriormente, se ha probado cómo actualizando el conjunto de referencia sólo por diversidad se obtienen peores resultados que considerando sólo el criterio de calidad (Laguna y Martí 2003).

Además, otros aspectos clave del método de SS sobre los cuales se sigue estudiando y que permiten implementarlo según diversas alternativas son los siguientes:

- **Control de la diversidad** cuando se forma el conjunto de referencia: Para garantizar la diversidad cuando se seleccionan las $b/2$ soluciones de mayor calidad de entre las P soluciones generadas por el método generador se puede establecer un umbral de distancia entre estas soluciones de alta calidad, de forma que una solución candidata solo puedan entrar a formar parte del conjunto de referencia si la distancia mínima entre esta solución y las que ya están en el conjunto de referencia sea igual o mayor que ese umbral establecido.
- La incorporación del **uso de memoria** en el algoritmo:
  - Para generar soluciones: además de poder generar el conjunto de P soluciones diversas (paso 1 del algoritmo) de forma aleatoria, se puede generar desarrollando algún método que use la memoria basada en la frecuencia con que aparecen los distintos elementos en las soluciones con objeto de evitar la repetición de soluciones similares y favorecer

así la diversidad. Es importante destacar que desde sus orígenes, SS, se basa en obtener diversidad de un modo determinista pre-establecido en lugar de recurrir a la aleatoriedad.

– Para su uso en el método de mejora: Cuando se añaden estructuras de memoria al método de mejora, éste deja de ser un simple procedimiento de búsqueda local (procedimiento heurístico) y se transforma en una estrategia metaheurística, es decir, una estrategia maestra que guía y modifica otras heurísticas para producir soluciones más allá de aquellas que normalmente se generan en una búsqueda de óptimos locales (Glover y Laguna 1997). El resultado es un método híbrido que combina dos estrategias metaheurísticas: Scatter Search y el método Tabu Search usado para mejorar las soluciones.

- Búsqueda de un **equilibrio entre diversificación e intensificación**: se trata de estudiar cómo se va a distribuir el tiempo total de computación, es decir, qué porcentaje del tiempo se dedica a generar las soluciones y cuál es el que se dedica a combinarlas.

- Buscar el **tamaño óptimo** del conjunto de referencia: Se debe estudiar si b debe ser un número pequeño tal y como se suele aconsejar o si por el contrario se obtendrían mejores resultados aumentando este número. Asimismo existe la posibilidad de considerar que b varíe en función del estado de la búsqueda.

- Probar con diversos **métodos de combinación**: En Campos y otros (2005) se analizan distintos métodos de combinación de soluciones, algunos con elementos aleatorios y otros deterministas, de forma que el algoritmo selecciona el método de combinación probabilísticamente, de acuerdo con los éxitos obtenidos por éste.

- Determinar a qué soluciones se debe aplicar el **método de mejora**: La aplicación del método de mejora a todas las soluciones generadas y combinadas no garantiza mejores resultados. En este sentido según Ugray y otros (2001) sería conveniente establecer umbrales de calidad para excluir de la aplicación del método de mejora a aquellas soluciones que difícilmente puedan llegar a ser la mejor solución.

- Formar el **conjunto de referencia inicial** con la mitad de soluciones obtenidas según el criterio de calidad y la otra mitad según el de diversidad. Se han hecho pruebas considerando otras proporciones (véase Laguna y Martí, 2003) pero en principio parece ser que ésta es la que permite obtener mejores resultados.

4 Aplicación de SS a un Problema de Localización

Scatter search se ha aplicado a la resolución de una gran variedad de problemas de optimización. En este caso concreto vamos a usar un SS propuesto en Pacheco y Casado (2005) para resolver un problema de localización, concretamente el problema de los p-centros.

El problema de los p-centros es un problema de localización bien conocido que consiste en colocar p servicios como colegios, hospitales o similar y asignar clientes a dichos servicios de forma que se minimice la máxima distancia entre un cliente y su servicio. Se trata de un problema NP-hard tal y como se demostró en Kariv y Hakimi (1979).

Sea $U = \{u_1, u_2, \ldots, u_m\}$ un conjunto de usuarios y $V = \{v_1, v_2, \ldots, v_n\}$ un conjunto de localizaciones donde colocar servicios. Considérese conocida la distancia d_{ij} entre cada cliente u_i y la localización v_j, el problema consiste en encontrar un subconjunto X de p localizaciones de forma que se minimice

$$\max_{i=1,\ldots,m} \left\{ \min_{v_j \in X} d_{ij} \right\}$$

El problema se puede formular de forma lineal como sigue

$$\begin{array}{lllll}
\text{Minimizar} & z & & & \\
\text{sujeto a:} & \sum_{j=1,\ldots,n} x_{ij} = 1, & i = 1 \ldots m; & & (1) \\
& x_{ij} \leq y_j, & i = 1 \ldots m; j = 1 \ldots n; & & (2) \\
& \sum_{j=1,\ldots,n} y_j = p; & & & (3) \\
& \sum_{j=1,\ldots,n} d_{ij} x_{ij} \leq z & i = 1 \ldots m; & & (4) \\
& x_{ij}, y_j \in \{0,1\} & i = 1 \ldots m; j = 1 \ldots n; & & (5)
\end{array}$$

donde $y_j = 1$ indica que se ha colocado un servicio en v_j (0 en caso contrario); $x_{ij} = 1$ indica que al usuario u_i se le ha asignado el servicio v_j (0 en caso contrario). Este modelo es usado por ejemplo en localización de estaciones de bomberos, policía o ambulancias, unidades de urgencias, etc.

Para resolver este problema de localización se ha diseñado una versión que podríamos denominar 'estática' de Scatter Search. La descripción en seudocódigo de forma general es la siguiente:

Algoritmo Scatter Search "estático"

1. Generar un conjunto inicial de P soluciones con un método Generador-Diversificador
2. Mejorar estas soluciones con un método de mejora

3. Con estas soluciones construir el conjunto de referencia inicial
4. Repetir
   - 4.1 Obtener todos los subconjuntos de pares del conjunto de referencia
   - 4.2 Combinar estos subconjuntos para obtener nuevas soluciones
   - 4.3 Mejorar estas nuevas soluciones con el método de mejora
   - 4.4 Actualizar el conjunto de referencia con estas nuevas soluciones hasta que se estabilice (i.e. no se incluyan nuevas soluciones)
5. Si han trascurrido max_iter iteraciones (pasos 1-4) sin mejora finalizar; sino volver al paso 1 (Reiniciar)

Denotamos por X la solución, total o parcial, en cada momento, es decir, las localizaciones (índices) donde se han colocado servicios, y f el valor de la función objetivo correspondiente a X.

Para formar el conjunto de Referencia, (paso 3) se comienza seleccionando las $b/2$ soluciones de mayor calidad según la función objetivo. A este número de soluciones de mayor calidad que forman el conjunto de Referencia, $b/2$, le denominamos *Tam_Ref1.* Posteriormente para añadir la otra mitad de soluciones se usa la siguiente función o criterio que mide la 'diversidad' de una solución candidata X a entrar con respecto a los que ya están en Refset

$$Difmin(X, Refset) = min\{dif(X, X') \mid X' \in Refset\};$$

donde $dif(X, X') = |X - X'|$, es decir el número de elementos (localizaciones) de la solución X que no están en X'.

La actualización de Refset (paso 4.4.) se realiza considerando la calidad de las soluciones. Es decir, se incorporan aquellas nuevas soluciones que mejoren la función objetivo de alguna de las soluciones existentes en Refset. A continuación se describen el método Generador-Diversificador, el método de mejora y el de combinación.

4.1 Método Generador-Diversificador

Nuestro método diversificador está basado en constructivos tipo GRASP. GRASP (greedy randomized adaptive search procedure), es un método heurístico que construye soluciones usando una función voraz y aleatoriedad controlada. La mayoría de las implementaciones GRASP incluyen un procedimiento de búsqueda local para mejorar las soluciones generadas. GRASP fue originalmente propuesto en el contexto de problemas de cubrimientos de conjuntos (Feo y Resende 1989).

Un tutorial clásico se puede encontrar en Feo y Resende (1995) y más recientemente en Pitsoulis y Resende (2002).

En nuestro caso la evaluación proporcionada por la función voraz Δ_j en cada paso es el valor de la función objetivo que se obtendría si se añadiera un servicio a j. El método diversificador consta de los siguientes pasos:

Procedimiento Avido-Aleatorio
Hacer $X = \emptyset$
Mientras $|X| < p$ hacer

- Determinar $\forall j \in V \setminus X$ el valor Δ_j de f si se añadiera j a X
- Determinar $\Delta max = max\{\Delta_j \mid j \in V \setminus L\}$ y $\Delta min = min\{\Delta_j \mid j \in V \setminus L\}$
- Construir $L = \{j \in V \setminus L \mid \Delta_j \leq \alpha \cdot \Delta min + (1 - \alpha) \cdot \Delta max\}$
- Elegir $j^* \in L$ aleatoriamente
- Hacer $X = X \cup \{j^*\}$

El parámetro α ($0 \leq \alpha \leq 1$) controla el nivel de aleatoriedad. A mayor valor de α menor nivel de aleatoriedad. Con este uso de aleatoriedad controlada se consigue una muestra de soluciones en la que normalmente la mejor de ellas supera a la encontrada con una elección totalmente determinística, (con $\alpha = 1$). Una selección adecuada de α permite un equilibrio entre diversificación y calidad de las soluciones.

La primera vez que se emplea el método generador-diversificador (paso 1) no hay 'historia' acerca de cuántas veces un elemento ha formado parte de las soluciones del conjunto de referencia. Sin embargo, esta información puede ser utilizable cuando el método se usa para reiniciar el proceso. La información se registra en el siguiente vector

$freq(j) =$ Número de veces que cada localidad j de V ha pertenecido a las soluciones del conjunto de referencia

La información registrada en $freq(j)$ se usa para modificar los valores Δ_j en el método diversificador de la siguiente manera

$$\Delta'_j = \Delta_j - \beta \Delta_{max} \frac{freq(j)}{freq_{max}}$$

donde $freqmax = \max\{freq(j) : \forall j\}$. Con los valores modificados Δ'_j se calculan Δ'_{min} y Δ'_{max} y se ejecuta el método diversificador con estos valores para construir

la lista de candidatos L. Obsérvese que con $\beta = 0$, el método diversificador modificado coincide con el original. Altos valores de β fuerzan a la selección de elementos que menos han aparecido. El uso de información dada por la frecuencia en el método diversificador está inspirada en Campos y otros (2005). Hay que destacar que la incorporación de memoria a la construcción, hace que ésta ya no sea de tipo GRASP en sentido estricto, pues el muestreo del espacio de soluciones ya no es aleatorio e independiente.

4.2 Método de Mejora

El método de mejora usado tiene su origen en Mladenovic y otros (2001) donde se hacen adaptaciones al problema de los p-centros de 3 heurísticos clásicos para el problema de las p-medianas tomados de Mulvey and Beck (1984). Uno de estos tres heurísticos, el procedimiento "Alternate", es el que hemos seleccionado como método de mejora y se describe a continuación:

Procedimiento Alternate
Repetir

- Para cada servicio j de X, determinar el subconjunto de los puntos de U que tienen a j como servicio más cercano
- Para cada uno de estos subconjuntos de usuarios resolver el problema del 1-centro
- Hacer X' el conjunto de soluciones de estos p problemas, y f' su valor
- Si $f' < f$ hacer $X = X'$ y $f = f'$

hasta que no haya cambios en X

4.3 Método de Combinación

e obtienen nuevas soluciones combinando pares del conjunto de referencia (paso 4.2). El número de soluciones generadas de cada par depende de la relativa calidad de las soluciones que son combinadas. Considérese x^t y x^q las soluciones del conjunto de referencia que son combinadas, donde $t < q$. Se asume que el conjunto de referencia está ordenado de forma que x^1 es la mejor solución y x^b la peor; entonces el número de soluciones generadas de cada combinación es:

tres si $t \leq Tam\_Ref1$ y $q \leq Tam\_Ref1$
dos si $t \leq Tam\_Ref1$ y $q > Tam\_Ref1$
uno si $t > Tam\_Ref1$ y $q > Tam\_Ref1$.

Cada par de soluciones del conjunto de referencia se usa para generar nuevas soluciones. Para ello se usa la estrategia denominada Path Relinking. La idea básica es construir un camino que una las dos soluciones. Algunas de las soluciones intermedias en dicho camino se utilizan como puntos iniciales a los que se les aplica la fase de mejora tal y como muestra la Figura 3.

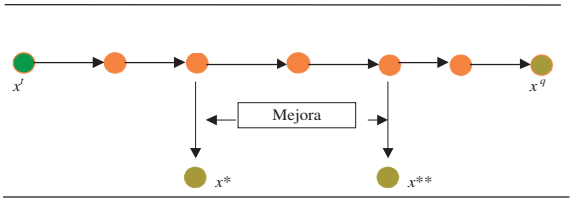


Fig. 3.- Generación de nuevas soluciones usando Path Relinking

El camino que une dos soluciones dadas, x^t y x^q, se construye como sigue. Inicialmente se hace $x = x^t$. En los siguientes pasos se añade a x un elemento de x^q que no esté en x y se elimina un elemento que no esté en x^q. De esta forma la solución intermedia x en cada paso tiene un elemento más en común con x^q . En cada paso se elige el mejor entre estos posibles cambios. Path Relinking es una estrategia tradicionalmente asociada a Tabu Search. La idea que subyace es que en el camino entre dos buenas soluciones, se espera que haya soluciones de parecida calidad (incluso en algún caso mejor). Para una mayor ilustración ver Glover, Laguna y Martí (2000).

4.4 Resultados Computacionales

Inicialmente, para mostrar el funcionamiento de SS se han hecho una serie de pruebas usando los ejemplos de la librería OR-Lib correspondientes a valores de $p \leq 10$. Los valores de los parámetros que ha usado SS en este caso son: $P = 12$, $b = 6$, $\alpha = \beta = 0,8$ y $max\_iter = 5$. En estos ejemplos $U = V$, es decir las localizaciones donde colocar las facilidades coinciden con los usuarios. En la Tabla 1 se muestran los resultados.

Además, se han hecho pruebas con problemas reales. Los datos de los problemas reales se refieren a diversas provincias en el norte de España, concretamente vila, León, Salamanca, Segovia y Burgos. Con estas experiencias se quiere analizar para cada provincia dónde situar una serie de unidades de diabetes entre las diferentes localidades que pueden acoger las mismas (por tener algún tipo de instalación que pueda considerarse adecuada). Los valores de p considerados son siempre menores o iguales a 10. Esto se debe a que las autoridades sanitarias correspondientes establecieron que, por diversas restricciones presupuestarias, el número máximo de unidades de diabetes que se podían abrir era diez.

En cada caso se ha considerado la matriz de tiempos (en minutos) entre todas las poblaciones origen, y las poblaciones que, potencialmente, pueden ser destinos.

n	p	$Scatter$ $Search$	$TiempoMejor$ $Solucion$
100	53	121	0,98
100	10	98	1,16
100	10	93	1,25
200	5	82	4,39
200	10	63	4,75
300	5	57	6,88
300	10	49	10,69
400	5	45	10,7
400	10	39	16,64
500	5	40	16,91
500	10	37	27,2
600	5	38	23,42
600	10	32	34,78
700	5	30	31,77
700	10	28	45,52
800	5	39	60,94
800	10	27	76,11
900	5	28	54,86
900	10	24	73,91

Tabla 1: *Resultados para ejemplos de OR-Lib, con* $p \leq 10$

Para hallar estos tiempos de recorrido se ha usado la información sobre carreteras suministrada por el CNIG (Centro Nacional de Información Geográfica), considerando diferentes velocidades según el tipo de tramo (Nacionales, Autonómicas, Provinciales etc. . .). Con esta información sobre la red de carreteras se ha calculado la matriz de tiempos usando el conocido algoritmo de Djikstra.

Estos problemas reales se han resuelto, además de con el SS descrito anteriormente, con un algoritmo basado en la estrategia metaheurística Búsqueda en Entornos Variables (VNS) tomado de Mladenovic y otros (2001). El criterio de parada que se ha considerado para ambas estrategias es un tiempo de computación máximo de 400 segundos. En estos ejemplos $U \neq V$ y los valores de m y n para cada provincia se muestran en la Tabla 2 junto con los resultados que se obtienen.

Si observamos los resultados obtenidos por ambos algoritmos vemos como aunque VNS obtiene en prácticamente todos los casos las mismas soluciones que SS, éste último alcanza sus soluciones en un tiempo considerablemente inferior al que emplea VNS.

	M	n	P	VNS	$Tiempo$ $mejorsol$	SS	$Tiempo$ $mejorsol$
vila	248	156	5	36	4,12	36	0,08
	248	156	10	25	106,4	23	3,72
Burgos	452	152	5	61	0,22	61	0,05
	452	152	10	41	7,20	41	0,05
León	211	184	5	47	13,4	47	2,84
	211	184	10	33	4,64	33	4,44
Salamanca	362	150	5	45	0,8	45	0,36
	362	150	10	31	227,68	31	5,68
Segovia	209	119	5	31	35,4	31	1
	209	119	10	22	87,88	22	2

Tabla 2: *Resultados para ejemplos de diversas provincias del norte de España* ($p \leq 10$)

5 Conclusiones

La Búsqueda Dispersa es una estrategia metaheurística que tiene su origen en los años setenta y se ha aplicado con éxito a la resolución de numerosos problemas de optimización. Aunque se trate de un método evolutivo presenta diferencias respecto a los algoritmos genéticos, probablemente el representante más conocido y extendido de los algoritmos evolutivos. En este trabajo se introducen los aspectos básicos de Búsqueda Dispersa, así como las múltiples alternativas que ofrece para explotar sus ideas fundamentales. Además se muestra una aplicación a la resolución de un conocido problema de localización.

Agradecimientos

Los autores de este trabajo agradecen la ayuda del Ministerio de Educación y Ciencia por la subvención económica para la realización de este trabajo a través del Plan Nacional de I+D, (Proyecto SEJ- 2005 08923/ECON), así como la recibida de la Conserjería de Educación de Castilla y León (Proyecto BU008A06).

6 Bibliografía

[1] Campos V., Glover F., Laguna M. y Martí R. An experimental Evaluation of Scatter Search for the Linear Ordering Problem. Journal of Global Optimization Vol. 21, pp 397-414. 2001.

[2] Campos V., Laguna M. y Martí R. Context-Independent Scatter and Tabu

Search for Permutation Problems, INFORMS Journal on Computing, Vol. 17(1), pp 111-122. 2005.

[3] Feo, T.A. y Resende, M.G.C. A Probabilistic heuristic for a computationally difficult Set Covering Problem. Operations Research Letters, Vol. 8, pp 67-71. 1989.

[4] Feo, T.A. y Resende, M.G.C. Greedy Randomized Adaptive Search Procedures. Journal of Global Optimization, vol. 2, pp 1-27. 1995.

[5] Glover F. Parametric Combinations of Local Job Shop Rules. Chapter IV. ONR Research Memorandum nº. 117, GSIA, Carnegie Mellon Universiy, Pittsburgh, PA. 1963.

[6] Glover F. A Multiphase Dual Agorithm for the Zero-One Integer Programming Problem. Operations Research, Vol. 13, nº 6, pp 879-919. 1965.

[7] Glover F. Surrogate Constraints. Operations Research, vol. 16, pp 741-749. 1968.

[8] Glover F. Heuristics for Integer Programming Using Surrogate Constraints. Decision Sciences, Vol. 8, pp 156-166. 1977.

[9] Glover F. Genetic Algorithms and Scatter Search: Unsuspected Potencials. Statistics and Computing, Vol. 4, pp 131-140. 1994.

[10] Glover F. A Template for Scatter Search and Path Relinking. J.k. Hao, E. Lutton, E. Ronald, M. Schoenauer, D. Syners (eds.). Artificial Evolution. Springer LNCS 1363, pp 13-64. 1998.

[11] Glover F. y Laguna M. Tabu Search. Kluwer Academic Publishers, Boston. 1997.

[12] Glover, F., Laguna, M. y Martí, R. Fundamentals of Scatter Search and Path Relinking. Control and Cybernetics, Vol. 39, no. 3, pp 653- 684. 2000.

[13] Kariv O. y Hakami S.L. An algorithmic approach to network location problems. Part I: The p-center Problem. SIAM J. Appl. Math., 37, pp 513-538. 1979.

[14] Laguna M. y Martí R. Scatter Search: Metodology and implementations in C. Kluwer Academia Publishers, Bostón. 2003.

[15] Mladenovic N., Labbe M. y Hansen P. Solving the p-center Problem with Tabu Search and Variable Neigborhood Search. Les Cahiers du GERAD, G-2000-35. 2001.

[16] Mulvey J.M. y Beck M.P. Solving Capacitated Clustering Problems. European Journal Operations Research, 18 (3), pp 339-348. 1984.

[17] Pacheco J. y Casado S. Solving Two Location Models with Few Facilities by Using a Hybrid Heuristic: a Real Health Resources Case, Computers and Operations Research 32, pp 3075-3091. 2005.

[18] Pitsolulis L.S. y Resende M.G.C. Greedy Randomized Adaptive Search Procedures. In Handbook of Applied Optimization, P. M. Pardalos and M. G. C. Resende (Eds.), Oxford University Press, pp 168-182. 2002.

[19] Ugray Z., Lasdon, L., Plummer J., Glover F, Kelly J. y Marti R. A Multistar Scatter Search Heuristic for Smooth NLP and MINLP Problems, to appear in INFORMS Journal on Computing. 2001.

Metaheurísticos en Programación Multiobjetivo*

R. Caballero[a], **J. Molina**[a], **A. G. Hernández-Díaz**[b]

[a] Universidad de Málaga,
Departamento de Economía Aplicada (Matemáticas)
[b] Universidad Pablo de Olavide, Sevilla,
Dpto de Economía, Métodos Cuantitativos e Historia Económica

1 Introducción

A la hora de encarar la resolución de un gran número de problemas de optimización correspondientes a situaciones reales, los métodos clásicos de resolución se encuentran muy a menudo con grandes dificultades que impiden abordar con garantías su resolución. Así, encontramos numerosas aplicaciones reales en campos como la Economía, la Ingeniería y la ciencia donde con métodos con un amplio soporte teórico matemático (que asegura la obtención de una solución óptima en condiciones "ideales") no se puede esperar obtener una solución o no se puede esperar obtener esta solución en un horizonte temporal razonable. Como ejemplo, podemos encontrar en Garey y Johnson [27] problemas cuyos tiempos estimados de resolución son del orden de $2 \cdot 10^8$ siglos, o numerosos problemas de optimización combinatoria donde siquiera encontrar un punto factible con un método tradicional es una tarea intratable.

*Los autores desean agradecer a los editores del presente volumen su amable invitación a colaborar con el presente trabajo. Asímismo, desean agradecer a los evaluadores anónimos su aportación a la mejora del presente trabajo. Este trabajo se ha realizado bajo la financiación del Ministerio de Educación y Ciencia, y de la Consejería de Innovación, Ciencia y Empresa de la Junta de Andalucía.

Por este motivo, a finales de los 50, Simon y Newell [63], y principios de los 60, Glover [28], comenzaron a aparecer una serie de métodos que proporcionaban en un tiempo computacional razonable soluciones factibles con un valor cercano al óptimo. Este tipo de estrategias se denominaron métodos heurísticos (del griego heuriskein: encontrar o descubrir). Así, Zanakis y Evans [74] definen un heurístico como "un procedimiento simple, a menudo basado en el sentido común, que ofrece una buena solución (aunque no necesariamente la óptima) a problemas difíciles de un modo fácil y rápido". Enseguida comenzaron a utilizarse ampliamente este tipo de métodos para la resolución de problemas hasta el momento intratables o de resolución demasiado costosa, tal y como se puede observar en el trabajo de Zanakis y otros [75] donde clasifican 442 artículos en revistas internacionales relacionados con heurísticos en el periodo 1975-1986. El éxito de este tipo de estrategias produjo un enorme interés en su estudio y desarrollo dando lugar a su evolución en los métodos metaheurísticos. Así, el término metaheurístico fue introducido por Glover en [29] para definir una estrategia superior que guía y modifica otros heurísticos para obtener mejores soluciones que las que obtendrían estos heurísticos en solitario. El éxito de este tipo de técnicas fue inmediato, como se puede observar en el informe del Committee on the Next Decade of Operations Research (CONDOR 1988), donde tres de los más utilizados metaheurísticos en la literatura, el Temple Simulado, los Algoritmos Genéticos y la Búsqueda Tabú, fueron evaluados como "extremadamente prometedores". El desarrollo de los metaheurísticos en los últimos diez a nos ha sido enorme, dando lugar incluso a la aparición de revistas especializadas como Journal of Heuristics (Kluwer Academic Press) o Evolutionary Computation (MIT Press). En su forma moderna este tipo de técnicas engloba una enorme variedad de lo que se considera búsquedas inteligentes. Sin embargo, una clasificación rigurosa de los metaheurísticos no es una tarea sencilla.

Una de las áreas de investigación emergentes en la cual los Algoritmos Metaheurísticos están alcanzando mayor popularidad es en el campo la Optimización Multiobjetivo. En un problema de optimización multiobjetivo tenemos dos o más funciones objetivo a optimizar simultáneamente en lugar de una sola. Como consecuencia, los problemas de optimización multiobjetivo no tienen en general una única solución sino todo un conjunto de soluciones que reflejan el trade-off existente entre dichos objetivos. El desarrollo de las diversas técnicas de Programación Multiobjetivo ha conducido a su utilización para la resolución de un gran número de casos reales donde, ademas de las dificultades habituales al enfrentarse a un problema real, como es la complejidad del modelo, en lo relativo a tamaño del problema (número de variables, restricciones...), o a la naturaleza de las funciones (no linealidad, no diferenciabilidad...) o de las variables (enteras, binarias...), nos encontramos con la dificultad adicional que representa la multiplicidad de objetivos y el hecho de que hemos de encontrar un conjunto de soluciones y no un único punto. Por estos motivos, aunque el enfoque multiobjetivo aplicado

sea correcto desde el punto de vista teórico, una implementación computacional puede no resultar válida algorítmicamente por carecer de herramientas para resolver el problema. Por estos motivos, resulta obvia la potencialidad del empleo de técnicas metaheurísticas en combinación con los diversos enfoques de programación multiobjetivo, que permitan abordar con garantías un amplio abanico de problemas reales. En este trabajo intentaremos recorrer los diferentes enfoques en la literatura a la hora de abordar este tipo de problemas mediante algoritmos metaheurísticos.

Así, en la siguiente sección comenzamos definiendo formalmente lo que se entiende por un problema de optimización multiobjetivo y otras definiciones asociadas. En la Sección 3 clasificamos los principales metaheurísticos para concluir en la Sección 4 con algunos comentarios sobre las aplicaciones reales de este tipo de técnicas, las lineas futuras y algunas conclusiones.

2 Problemas Multiobjetivo

Entendemos por Problema de Optimización Multiobjetivo (MOP) a aquellos de la forma:

$$minimizar\ [f_1(x), f_2(x), ..., f_k(x)]$$

sujeto a m restricciones de desigualdad:

$$g_i(x) \leq 0, \quad i = 1, 2, ..., m$$

y p restricciones de igualdad:

$$h_i(x) = 0, \quad i = 1, 2, ..., p$$

donde k es el número de funciones objetivo y n el número de variables de decisión del problema. Así, denotaremos por $x = (x_1, x_2, ..., x_n)^t$ al vector de variables de decisión.

Normalmente, los objetivos del problema están en conflicto unos con otros y por ello raramente se da el caso en el que un solo vector optimiza simultáneamente a todas las funciones objetivo (en este caso además el problema no sería realmente multiobjetivo). Por tanto, la definición de optimalidad es diferente a la usual de los problemas mono-objetivo. La noción de optimalidad comúnmente aceptada para problemas multiobjetivo es la conocida como *optimalidad de Pareto* [54]:

Diremos que un vector de variables de decisión $x^* \in \mathcal{F}$ es un *óptimo de Pareto* si no existe otro $x \in \mathcal{F}$ tal que $f_i(x) \leq f_i(x^*)$ para todo $i = 1, ..., k$ y $f_j(x) < f_j(x^*)$ para al menos un j.

En otras palabras, x^* es un óptimo de Pareto si no existe otro punto factible $x \in \mathcal{F}$ que mejore alguno de los objetivos sin causar simultáneamente un empeoramiento en alguno de los otros criterios. Ahora bien, este concepto casi siempre

nos proporciona más de una solución. A este conjunto se le denomina *conjunto de óptimos de Pareto*. A los vectores x^* incluidos en el conjunto de óptimos de Pareto también se les conoce como *soluciones no dominadas*. A la imagen del conjunto de óptimos de Pareto por las funciones objetivo se le denomina *frontera de Pareto*. Por tanto, el objetivo es determinar de entre todos los elementos del conjunto $\mathcal{F}$, formado por todos los vectores que satisfagan las restricciones, aquellos que constituyen soluciones no dominadas.

Existen en la literatura diferentes procedimientos para convertir un problema multiobjetivo en uno mono-objetivo. Quizás el procedimiento más sencillo es el uso de combinaciones de éstos (usualmente mediante combinaciones lineales) para su transformación en un único objetivo. Estas técnicas son conocidas como *funciones de agregación* ya que se combinan o agregan todos los objetivos en uno solo. Un ejemplo de este procedimiento es mediante la combinación lineal

$$min \sum_{i=1}^{k} \omega_i f_i(x)$$

donde $\omega_i \geq 0$ son los pesos que representan las importancia relativa de cada uno de los k objetivos de nuestro problema. Es usual suponer que

$$\sum_{i=0}^{k} \omega_i = 1.$$

Las funciones de agregación pueden ser lineales (como en el ejemplo anterior) o no lineales (cuando por ejemplo se utilizan distancias de tipo no-lineal). Ambos tipos de funciones de agregación han sido utilizadas con algoritmos metaheurísticos, obteniéndose un éxito relativo.

Las funciones de agregación han sido en gran parte subestimadas por los investigadores debido a las limitaciones que poseen principalmente las funciones de agregación lineales (no pueden generar porciones no convexas de la frontera de Pareto a pesar del uso de combinaciones de pesos [15]). No obstante, las funciones de agregación no lineales no necesariamente presentan esta limitación [8]. De hecho, se pueden definir funciones de agregación lineales con cierto ingenio capaces de generar fronteras de Pareto no convexas. No obstante, la comunidad de metaheurísticos tiende a prestar cada vez menos interés en las funciones de agregación.

Por otra parte, existe una gran variedad de técnicas multiobjetivo representando distintas filosofías, que no recurren a la agregación mediante pesos de las funciones objetivo, como es la Programación por Metas, el método de la ϵ-restricción o la Programación Compromiso. Una descripción detallada de este tipo de técnicas puede encontrarse en [65].

3 Algoritmos Metaheurísticos

La gran cantidad de metaheurísticos existentes hoy en día para el tratamiento y la resolución de problemas de optimización multiobjetivo complican enormemente la tarea de clasificarlos. No obstante, a grandes rasgos, podemos afirmar que existen dos grandes grupos de heurísticas: los basados en búsquedas por entornos, especialmente diseñados para búsquedas locales, y los basados en poblaciones, cuyo representante más destacado son los Algoritmos Genéticos.

No obstante, hibridizaciones de estas técnicas y otras más recientes inspiradas en la Biología o Ciencias de la Naturaleza están siendo también aplicadas con gran éxito. A continuación, describimos brevemente el funcionamiento de las principales heurísticas.

3.1 Basados en Búsquedas por Entornos

Estos métodos tienen en común utilizar una operación básica que denominamos *movimiento* que consiste en la modificación de características o elementos de una solución para crear una serie de soluciones posibles que constituyen un vecindario de dicha solución, y de entre las cuales se elegirá un elemento para pasar a la siguiente iteración. Destacamos los siguientes:

- **Temple Simulado**: En 1983, Kirkpatrick et alt. [47] introdujeron el concepto de Temple Simulado para la optimización combinatoria. El nombre de Temple Simulado se debe a las similitudes que presenta con el proceso físico conocido como temple, en el cual un material es calentado hasta el estado líquido y luego es enfriado lentamente para obtener un refinamiento del mismo. La principal virtud de este método es su capacidad para escapar de los óptimos locales, lo cual hace que sea ampliamente utilizado en aplicaciones prácticas en la literatura. Así, en el trabajo de Vidal [72] podemos encontrar por ejemplo aplicaciones al Problema del Viajante, al Problema de Rutas de Vehículos o al dise no de redes de telecomunicaciones. El Temple Simulado fue utilizado para la Programación Multiobjetivo por primera vez en 1992 en el trabajo de Serafini [62], donde la principal idea para la adaptación de este método para problemas con criterios múltiples es la utilización de un criterio de aceptación de soluciones de peor calidad basado en la relación de dominancia entre dos soluciones dadas. Otros trabajos en los que se ha utilizado Temple Simulado para la Programación Multiobjetivo son Ulungu et alt. [68, 69, 70], Teghem et alt. [67] (donde la principal idea de estos trabajos es utilizar funciones agregativas basadas en pesos e ir variando adeacuadamente estos pesos) o Czyzak y Jaszkiewicz [13] (donde la gestión de estos pesos se hace de forma dinámica intentando moverse en direcciones no exploradas previamente). Las adaptaciones del Temple

Simulado a problemas multiobjetivo se han mostrado en general muy eficientes y muy robustas, lo cual lo ha hecho muy popular para la resolución de problemas reales. Así, en el trabajo de Hapke et alt. [39] podemos encontrar una aplicación del Temple Simulado al Multicriteria Project Scheduling Problem. Estos mismos autores en su trabajo de 2000 [40] vuelven a aplicar con éxito el Temple Simulado a problemas de Programación Multiobjetivo Difusa. Ponce y Matos [56] utilizan el Temple Simulado para resolver problemas multiobjetivo de distribución y planificación de redes. Viana y Pino da Sousa [71] utilizan un Temple Simulado para resolver problemas multiobjetivo de planificación de proyectos. Más detalles acerca de trabajos relacionados con Temple Simulado y Programación Multiobjetivo pueden encontrarse en Erhgott y Gandibleux [18] y en Jones et alt. [45].

- **Búsqueda Tabú**: La búsqueda tabú tiene sus orígenes en un trabajo de Glover de 1986 [29]. La búsqueda tabú es el principal metaheurísitco dentro de lo que se conoce como Programación mediante Memoria Adaptativa (AMP), que se caracterizan por ser métodos de búsqueda por entornos en los cuales se utiliza información acerca de los movimientos realizados con anterioridad. Los principales atributos de cada solución visitada son almacenados en una lista (lista tabú) durante un determinado número de iteraciones, para evitar que estas soluciones sean revisitadas, es decir, para evitar ciclos en la búsqueda por entornos. Un elemento del vecindario de la solución actual es declarado tabú (es decir, es prohibido) si alguno de sus atributos está en la lista tabú. Dentro de un método de búsqueda tabú encontramos una gran variedad de distintas estrategias destinadas a mejorar la búsqueda, como son, por ejemplo, la intensificación, que permite concentrar la búsqueda en aquellas zonas más prometedoras; la diversificación, que permite desplazarse hacía zonas no exploradas; la oscilación estratégica, que permite visitar zonas infactibles temporalmente; o el reencadenamiento de trayectorias, que permite combinar soluciones. Más detalles acerca de la búsqueda tabú pueden encontrarse en Glover y Laguna [31]. La búsqueda tabú es uno de los metaheuristicos más utilizados, tal y como se puede observar en [32] donde se citan 70 áreas diferentes donde ha sido aplicado, desde los problemas de rutas de vehículos, de distribución de energía eléctrica o dise no de redes de transporte. En cuanto a los métodos de búsqueda tabú para problemas multiobjetivo, los procedimientos encontrados en la literatura se centran en algún tipo de agregación de los criterios para transformar el problema en un problema mono-objetivo que será resuelto mediante la búsqueda tabú. Así, en Dahl et alt. [14] se genera una familia de vectores de pesos para cada uno de los cuales se resuelve el problema mono-objetivo resultante de agregar los criterios mediante una suma ponderada por este vector de pesos. En Hertz et alt. [38] se resuelve una serie de problemas

mono-objetivo considerando por turnos cada una de las funciones objetivo junto con una función de penalización. En Gandibleux et alt. [24] utilizan una función escalarizada de logro basada en las funciones objetivo del problema como guía para la búsqueda tabú. Otros trabajos en los que se encuentran enfoques de búsqueda tabú para problemas multiobjetivo son los de Hansen [36], Ben Abdelaziz et alt. [5], Gandibleux y Freville [26], Alves y Climaco [2], Al-Yamani y otros [3], Caballero et alt. [6] o Ehrgott et alt. [19].

- **GRASP**: Este método, que aparece por primera vez en el trabajo de Feo y Resende [21], recibe su nombre de las iniciales en inglés de Procedimiento de Búsqueda basado en funciones Ávidas, Adaptativas y Aleatorias.

  GRASP es un método iterativo en el que cada iteración aporta una solución al problema, siendo la solución final aportada la mejor de las soluciones encontradas. Cada iteración de un GRASP incluye dos fases: la primera construye inteligentemente una solución inicial a través de una función ávida, aleatoria y adaptativa; y la segunda aplica un procedimiento de búsqueda local a la solución construida con idea de encontrar una mejora para esta solución.

  Una de las ventajas de este método es su sencillez y facilidad de implementación, que ha hecho de él uno de los metaheurísticos de mayor desarrollo actualmente. En cuanto a su utilización para problemas multiobjetivo encontramos los trabajos de Gandibleux et alt. [25] y Higgins et alt. [41].

3.2 Basados en Poblaciones: Algoritmos Evolutivos

Los Algoritmos Evolutivos son heurísticos que utilizan mecanismos de selección natural como motor de búsqueda para resolver problemas. Además, una de sus principales ventajas por la que son tan utilizados actualmente es porque los Algoritmos Evolutivos son capaces de evolucionar a todo un conjunto de posibles soluciones (también llamada población) que nos permitirán encontrar varios miembros del conjunto de Pareto en una ejecución simple del algoritmo, en lugar de tener que hacer varias ejecuciones, como ocurría con las técnicas de programación matemática tradicionales. Además, los Algoritmos Evolutivos son menos sensibles a características geométricas de la frontera de Pareto como la concavidad, convexidad, continuidad, etc.

Concretamente, un Algoritmo Evolutivo es un proceso estocástico e iterativo que opera sobre un conjunto P de individuos donde cada uno de estos individuos contiene una serie de cromosomas que le permiten representar una solución. Cada individuo es evaluado a través de una función de adecuación, de forma que se predispone la selección de aquellos individuos con mejor valor de adecuación para

su reproducción. Dentro de los Algoritmos Evolutivos encontramos tres grandes familias: la Programación Evolutiva, las Estrategias de Evolución y los Algoritmos Genéticos, aunque en este capítulo nos centraremos únicamente en esta última familia por ser la técnica evolutiva más popular en la actualidad y por ser la familia que concentra las aplicaciones al campo de la Programación Multiobjetivo.

Los Algoritmos Genéticos se han convertido en una herramienta muy utilizada para optimización, aprendizaje de máquinas, problemas de dise no, problemas de redes neuronales y otros campos de la Matemática y la Ingeniería. En la naturaleza, los individuos han de adaptarse a su entorno para sobrevivir mediante el proceso que denominamos evolución, en el cual aquellos aspectos o cambios que favorecen su competitividad son preservados, y aquellos aspectos que debilitan su adaptación son eliminados. Estas características, favorables o desfavorables, se almacenan y controlan desde unas unidades llamadas genes, que a su vez se agrupan formando unos conjuntos llamados cromosomas. A finales de los 60, John Holland [43] se interesó en aplicar los principios de la evolución natural para la resolución de problemas complejos en el campo del aprendizaje de máquinas, dando lugar a lo que hoy se conoce como algoritmos genéticos. En 1989 Goldberg [33] publicó un libro en el cual se asentaba una sólida base científica para este tipo de estrategias y en el cual se incluían más de 70 aplicaciones reales con éxito de Algoritmos Genéticos. Los primeros intentos de utilizar múltiples criterios en un algoritmo genético se centraban fundamentalmente en el uso de funciones agregativas (para más detalles, véase la Sección 2). El Método de la Restricción es utilizado por Ritzel et alt. [58], dando lugar a mejores aproximaciones de la frontera eficiente, aunque con un coste computacional mayor debido a que se han de realizar múltiples resoluciones con distintas cotas para las restricciones y distintas funciones a optimizar. Wilson y Macleod [73] utilizan con éxito un enfoque de Programación por Metas Ponderadas para resolver un problema con múltiples objetivos. Sin embargo, desde entonces se han desarrollado otros enfoques distintos que se basan en la eficiencia de Pareto y en otros tipos de ordenación.

Para evitar las dificultades a la hora de agregar los criterios, muchos de los esfuerzos en la literatura se han dirigido hacia enfoques basados en rankings. Así en 1985 Schaffer [61] desarrolló el método VEGA (Vector Evaluated Genetic Algorithm), cuya única diferencia con un algoritmo genético usual es la forma en que se realiza la selección para la reproducción. En este caso, en cada generación la población se agrupa en un cierto número de subpoblaciones (tantas como criterios) atendiendo en cada una de ellas al valor de una de las funciones objetivo. Dentro de cada una de estas poblaciones se realiza la selección atendiendo al valor de este criterio y luego las poblaciones se mezclan de nuevo para aplicar el operador de cruce y el operador de mutación, dando lugar a la siguiente generación. VEGA tiene varios problemas entre los cuales destaca su incapacidad para retener buenas soluciones eficientes (aquellas que no son las mejores en ninguno de los objetivos pero consiguen un buen compromiso entre todos ellos). Aunque su

autor sugirió varias mejoras, su mayor desventaja sigue siendo el hecho de no incorporar directamente una selección que incorporase la dominancia de Pareto.

Por otra parte, Fourman [23] utilizó un orden lexicográfico para realizar la selección de los individuos a reproducirse, obteniendo muy buenos resultados puesto que el orden lexicográfico es un orden total y por tanto siempre se puede determinar cuándo un individuo está mejor adaptado que otro, lo cual permite realizar una selección por ranking.

Basados en la Optimalidad de Pareto

A continuación aparecieron en la literatura lo que se ha denomindo enfoques basados en el orden de Pareto. Con este tipo de enfoques, el valor de adaptación de cada individuo depende no del valor de cada uno de los criterios, sino de su eficiencia o dominación dentro de cada población. La idea es encontrar los individuos en cada generación que no están dominados por ningún otro individuo, asignarles el ranking más alto y extraerlos de la población. Con el resto de individuos se repite este proceso hasta que todos ellos tienen asignado una posición en el ranking según este proceso, realizándose entonces una selección por ranking. Este enfoque se ha mostrado superior al enfoque VEGA en algunos casos, como se puede ver en Hilliard et alt. [42]. Los algoritmos más representativos de la primera generación son:

1. NSGA: Srinivas y Deb [64] evolucionaron la idea del ranking de no-dominación para dar lugar al método NSGA (Non-Dominated Sorting Genetic Algorithm) que se ha mostrado también muy eficiente a la hora de resolver problemas con múltiples criterios. Antes de seleccionar a los individuos, los puntos son clasificados en función de su no-dominación, esto es, a todos los puntos no dominados de la población se les asigna un mismo valor de aptitud proporcional al tama no de la población. Una vez eliminados estos puntos, se repite el proceso hasta que todos los puntos de la población inicial estén clasificados. Este proceso tiene una doble finalidad. Por una lado, aquellos puntos con mayor valor de aptitud tiene una mayor probabilidad para reproducirse en la siguiente población (lo que garantizará seguir explorando en las zonas no dominadas) y, por otro lado, el hecho de que conjuntos de puntos comparten el mismo valor de la función de aptitud garantiza la diversidad en el conjunto de puntos eficientes.

2. NPGA: NPGA (Niched Pareto Genetic Algorithm) fue propuesto por Horn y Nafpliotis [44] en 1993. NPGA usa un esquema de selección por torneo basado en la dominancia de Pareto: dos individuos de la población son elegidos aleatoriamente y comparados contra un subconjunto de la población entera (normalmente el 10% de la población). Si alguno de ellos es dominado (por los individuos aleatoriamente seleccionados de la población) y

el otro no, entonces el individuo no dominado gana. Cuando ambos competidores son dominados o no dominados el resultado del torneo se decide a través de fitness sharing o método de proporción (se pretende lograr la formación de subconjuntos de elementos vecinos en la población llamados nichos, reduciendo la aptitud de los individuos por la presencia de otros muy parecidos, véase [60]).

3. MOGA: Fonseca y Fleming [22] profundizan en la idea del ranking de los individuos en su algoritmo MOGA (Multi Objective Genetic Algorithm). En dicho algoritmo, el ranking de cierto individuo corresponde al número de cromosomas en la población actual por los cuales es dominado. Consideremos por ejemplo un individuo x_i en la generación t que es dominado por $p_i^{(t)}$ individuos de la generación actual. Entonces, el ranking de x_i viene dado por $rank(x_i, t) = 1 + p_i^{(t)}$. Así, los individuos no dominados poseen ranking igual a 1 mientras que los dominados son penalizados en función de la densidad de individuos en sus correspondientes regiones de dominancia.

La segunda generación de los Algoritmos Evolutivos nace con la introducción de la noción de *elitismo*. En el contexto de la optimización multiobjetivo, el elitismo usualmente se refiere al uso de una población externa (también denominada población secundaria) para almacenar todos los individuos no dominados encontrados hasta el momento. No obstante, el uso de un fichero externo plantea diversas cuestiones tales como la manera en que interactúan las dos poblaciones o el tama no de la población secundaria.

El elitismo también puede ser introducido a través del uso de $(\mu + \lambda)$-selección en la cual los μ padres compiten con sus λ hijos y aquellos que son no dominados son seleccionados para la siguiente generación.

Los más representativos de la segunda generación son los siguientes:

1. SPEA: El algoritmo Strength Pareto Evolutionary Algorithm (SPEA) fué desarrollado por Zitzler y Thiele [76] en 1999 sobre la idea de mantener un archivo externo de soluciones no-dominadas encontradas hasta el momento, pero incorporando un proceso de tipo cluster (denominado average linkage method) sobre este archivo para favorecer la diversidad y reducir su tama no pero sin destruir sus características.

2. SPEA2: Posteriormente aparece SPEA2 [77] el cual posee principalmente tres diferencias con respecto a su predecesor: (1) incorpora una estrategia de asignación de aptitud de "grano fino" (para cada individuo se tiene en cuenta el número de individuos que domina y el número de individuos por los que es dominado); (2) utiliza una técnica para la estimación de la densidad de su vecindario que guía la búsqueda más eficientemente, y finalmente (3) incorpora un método para truncar el archivo garantizando el matenimiento de las soluciones extremas.

3. PAES: Knowles y Corne proponen en 2000 [48] el algoritmo denominado *Pareto Archived Evolution Strategy* (PAES), que incorpora una estrategia evolutiva del tipo $(1 + 1)$ (un solo padre genera a un solo hijo compitiendo entre ellos) junto con una búsqueda local y la utilización de un archivo histórico para almacenar las soluciones no dominadas que se van encontrando a lo largo de la ejecución del algoritmo. Para mantener la diversidad apropiada en este archivo histórico se introdujo la idea de una malla adaptativa que ofrece ciertas ventajas con respecto a la utilización de los nichos del NPGA.

4. NSGA-II: Deb et alt. [16] solventan muchos de los problemas de la versión original del algoritmo con el NSGA-II. NSGA-II es más eficiente (computacionalmente hablando), usa elitismo y un operador de comparación (operador de crowding) en función de la proximidad de soluciones alrededor de cada uno de los puntos de la población. NSGA-II no usa una población externa como su predecesor pero su mecanismo de selección ahora consiste en la combinación de los mejores padres con los mejores hijos obtenidos (selección del tipo $(\mu + \lambda)$).

5. NPGA2: Erickson et alt. [20] propusieron una versión revisada del NPGA. Incorporan ranking de Pareto pero mantienen la selección mediante torneo. En este caso, no utilizan memoria externa y utilizan un mecanismo de elitismo similar al adoptado por NSGA-II.

En general, un problema común a las técnicas de este grupo radica en el uso de un proceso de jerarquización basado en la dominancia de Pareto, siendo precisamente este proceso el que consume más tiempo en una ejecución ya que es un proceso de orden $O(kM^2)$ donde k es el número de funciones objetivo y M es el tama no de la población. Adicionalmente a esto, un mecanismo extra es requerido para preservar la diversidad de la población que, en las formas en que se realiza en la actualidad, implica el uso de procesos de orden $O(M^2)$.

3.3 Nuevas tendencias

Además de estos dos grandes grupos de metaheurísticos, basados en Búsqueda por Entornos y Basados en Población, existe actualmente una gran variedad de métodos combinando ambos enfoques e intentando adaptar otros tipos de metaheuristicos que se han mostrado muy eficientes en el campo de la optimización mono-objetivo. A continuación enumeramos algunos ejemplos.

Búsqueda Dispersa

La Búsqueda Dispersa es un metaheurístico introducido en los setenta en el trabajo de Glover [30] de 1977 para Programación Entera. Un procedimiento

de Búsqueda Dispersa está basado fundamentalmente en la combinación de las soluciones de un conjunto de referencia para construir a partir de estos elementos nuevas soluciones que mejoren los elementos de este conjunto. En este sentido se puede clasificar este método como un algoritmo evolutivo, puesto que el procedimiento fundamental para encontrar nuevas soluciones es la combinación de las soluciones existentes. Sin embargo, en un procedimiento de Búsqueda Dispersa esta combinación de soluciones se hace de una forma sistemática, sin componentes aleatorios, sobre un conjunto peque no de soluciones de refencia, lo cual representa una diferencia fundamental con los principios fundamentales de un algoritmo evolutivo en general, donde la componente aleatoria es fundamental y donde se recombinan un gran número de soluciones. En los últimos a nos este procedimiento ha sido aplicado con éxito a numerosos problemas complejos de Programación Matemática, lo cual ha producido un sensible aumento en el número de trabajos utilizando este tipo de metaheurístico, tal y como se puede apreciar en Laguna y Martí [49]. Sin embargo, hasta el momento las aplicaciones de la Búsqueda Dispersa al campo de la Programación Multiobjetivo han sido muy escasas. Destacamos los trabajos de Beausoleil [4] y Molina et alt. [51].

Colonias de Hormigas

Los algoritmos ACO (Ant Colony Optimization) son modelos inspirados en el comportamiento de colonias de hormigas reales. Estudios realizados explican cómo animales casi ciegos, como son las hormigas, son capaces de seguir la ruta más corta en su camino de ida y vuelta entre la colonia y una fuente de abastecimiento. Esto es debido a que las hormigas pueden "transmitirse" información entre ellas gracias a que cada una de ellas, al desplazarse, va dejando un rastro de una sustancia llamada feromona a lo largo del camino seguido. Así, mientras una hormiga aislada se mueve de forma esencialmente aleatoria, los "agentes" de una colonia de hormigas detectan el rastro de feromona dejado por otras hormigas y tienden a seguir dicho rastro. Éstas a su vez van dejando su propia feromona a lo largo del camino recorrido y por tanto lo hacen más atractivo, puesto que se ha reforzado el rastro de feromona. Sin embargo, la feromona también se va evaporando con el paso del tiempo provocando que el rastro de feromona sufra, por otro lado, cierto debilitamiento. En definitiva, puede decirse que el proceso se caracteriza por una retroalimentación positiva, en la que la probabilidad con la que una hormiga escoge un camino aumenta con el número de hormigas que previamente hayan elegido el mismo camino. El primer algoritmo basado en la optimización mediante colonias de hormigas fue aplicado al Problema del Viajante (Dorigo et alt. [17]), obteniéndose unos resultados bastante alentadores. A partir de dicho algoritmo se han desarrollado diversos heurísticos que incluyen varias mejoras, y han sido aplicados a problemas de rutas [11]. Este método también se ha adaptado con éxito a la Programación Multiobjetivo, tal y como puede verse

en Gravel et alt. [34] o en Guntsch y Middendorf [35], y constituye actualmente uno de los métodos mas prometedores para la resolución de problemas de rutas multiobjetivo.

Cúmulo de Partículas (PSO)

Los algoritmos de optimización mediante Cúmulo de Partículas (Particle Swarm Optimization, en adelante PSO) han sido propuestos recientemente por Kennedy y Eberhart [46] motivados por el comportamiento social de las bandadas de pájaros o los bancos de peces.

PSO, como herramienta de optimización, origina un algoritmo basado en población en el cual los individuos, denominados partículas, cambian su posición a lo largo del tiempo. En un PSO las partículas vuelan en el espacio de búsqueda multidimensional. Durante el vuelo cada partícula ajusta su posición de acuerdo a su propia experiencia y a la de las partículas más cercanas haciendo uso de la mejor posición encontrada por él y por sus vecinos. Así, se puede interpretar como un algoritmo híbrido entre un genético y un memético. Recientemente ha sido adaptado para problemas multiobjetivo en [59].

Evolución Diferencial

La Evolución Diferencial (DE) es una heurística relativamente reciente propuesta por Storn y Price [66] para problemas de optimización sobre dominios continuos. En una DE, cada variable de decisión se representa en el cromosoma por un número real (codificación real). Como en otros algoritmos evolutivos, la población inicial de una DE se genera aleatoriamente. Para el proceso de selección se seleccionan tres padres (uno de ellos se designará el *padre principal*) los cuales generarán un único hijo (en lugar de dos, como en muchos de los algoritmos genéticos) que competirá con el padre principal. Este hijo se genera sumando al padre principal la diferencia de los otros dos padres.

Podemos encontrar en la literatura distintas extensiones de la DE para problemas multiobjetivos. Destacamos, entre otros, PDE [10] (maneja una única población, para la reproducción toma únicamente soluciones no dominadas y se utiliza una métrica de distancia para favorecer a la diversidad), PDEA [50] (combina la DE con distintos elementos del NSGA-II), VEDE [55] (DE con múltiples poblaciones que maneja en paralelo inspirado en VEGA) y DEMORS [37] donde se combina la evolución diferencial con el uso de Rough Sets, una herramienta de Inteligencia Artificial.

Algoritmos Culturales

Los Algoritmos Culturales fueron desarrollados por Robert G. Reynolds [57] como un complemento a la metáfora que usan los algoritmos de Computación

Evolutiva, que se habían concentrado en conceptos genéticos, y de selección natural. Los Algoritmos Culturales están basados en las teorías de algunos sociólogos y arqueólogos que han tratado de modelar la evolución cultural. Tales investigadores indican que la evolución cultural puede ser vista como un proceso de herencia en dos niveles: el nivel micro-evolutivo, que consiste en el material genético heredado por los padres a sus descendientes, y el nivel macro-evolutivo, que es el conocimiento adquirido por los individuos a través de las generaciones, y que una vez codificado y almacenado, sirve para guiar el comportamiento de los individuos que pertenecen a una población. Reynolds intenta captar ese fenómeno de herencia doble en los Algoritmos Culturales. El objetivo es incrementar las tasas de aprendizaje o convergencia, y de esta manera, que el sistema responda mejor a un gran número de problemas.

Los Algoritmos Culturales operan en dos espacios. Primero, el espacio de la población, como en todos los métodos de Computación Evolutiva, en el que se tiene un conjunto de individuos. Cada individuo tiene un conjunto de características independientes de los otros, con las que es posible determinar su aptitud. A través del tiempo, tales individuos podrán ser reemplazados por algunos de sus descendientes, obtenidos a partir de un conjunto de operadores aplicados a la población. El segundo espacio es el de *creencias*, donde se almacenarán los conocimientos que han adquirido los individuos en generaciones anteriores. La información contenida en este espacio debe ser accesible a cualquier individuo, quien puede utilizarla para modificar su comportamiento. La mayoría de los pasos de un algoritmo cultural corresponden con los de los algoritmos tradicionales de Computación Evolutiva, y se puede apreciar que las diferencias están en los pasos que incluyen al espacio de creencias.

Sistema Inmune Artificial

El sistema inmune ha servido como inspiración para solucionar problemas complejos de ingeniería y la ciencia con gran éxito, debido principalmente a que es un sistema de aprendizaje distribuido con interesantes características. Una de las principales tareas del sistema inmune es mantener al organismo sano. Algunos microorganismos (llamados patógenos) que invaden al organismo pueden resultar dañinos para éste. Los antígenos son moléculas que se encuentran expresadas en la superficie de los patógenos que pueden ser reconocidos por el sistema inmune y que además son capaces de dar inicio a la respuesta inmune para eliminarlos. Esta respuesta defensiva del sistema inmune presenta interesantes características desde el punto de vista del procesamiento de información. Es por ello que se ha usado como inspiración para crear soluciones alternativas a problemas complejos de ingeniería y la ciencia. Esta es un área relativamente nueva a la cual se le llama Sistema Inmune Artificial [53].

Las primeros intentos por resolver problemas de optimización multiobjetivo

(con o sin restricciones) usando un sistema inmune artificial basado en el concepto de optimalidad de Pareto se deben a Coello y Cruz-Cortés [9].

4 Conclusiones

La popularidad que en los últimos años han adquirido los Algoritmos Metaheurísticos se ha visto también reflejado en el número de trabajos publicados con aplicaciones a problemas multiobjetivo del mundo real (algo que no nos debe extra nar ya que la mayoría de los problemas reales son de naturaleza multiobjetivo). Prueba de ello es la reciente publicación del libro de Coello y Lamont [10] dedicado explícitamente a las aplicaciones de los Algoritmos Metaheurísticos y en especial de los Algoritmos Evolutivos a problemas de optimización multiobjetivo. Dichas aplicaciones pueden ser clasificadas en tres grandes grupos: ingeniería, industria y otras ciencias. Dentro de las aplicaciones a la ingeniería destacamos las aplicaciones a la ingeniería electrica, hidráulica, aeronáutica, robótica o control. Por otro lado, en el grupo de las aplicaciones en la industria destacamos el los problemas de dise no, logística, almacenamiento, distribución o scheduling. Finalmente, las aplicaciones en Física, Química, Medicina o Ciencias de la Computación destacan dentro del tercer grupo.

Hemos de señalar tambien el auge actual del campo de los Metaheuristicos para la Programación Multi-objetivo, tal y como refleja la sección de Nuevas Tendencias, donde se puede ver el esfuerzo en la literatura por extender al caso multiobjetivo un gran número de metaheuristicos diseñados para problemas monoobjetivo, constituyendo una garantía de desarrollo de numerosas lineas futuras.

Por otra parte, observamos en la inmensa mayoría de las adaptaciones actuales de metaheurísticas a la Programación Multiobjetivo tienen como finalidad la aproximación de la frontera eficiente, pero en la metodología multicriterio y en especial en las aplicaciones, es necesario encontrar una solución entre el gran número de soluciones generadas en la aproximación de la frontera eficiente. En este sentido, los Métodos Interactivos surgen de manera natural para la obtención final de soluciones adaptadas a las preferencias del decisor. Todos estos aspectos deben ser incorporados en los metaheurísticos implementados, y no son usuales en la bibliografía existente.

Otra carencia importante en el campo de los metaheuristicos para Programación Multiobjetivo es el desarrollo de métodos con capacidad eficiente para el manejo de restricciones. La mayor parte de los métodos encontrados en la literatura son desarrollados para problemas sin restricciones o con restricciones muy simples, tal y como se señala en Coello [7]. Sin embargo, no hace falta señalar que esto puede ser un problema a la hora de resolver problemas de ámbito real. Por este motivo, el uso de técnicas de manejo de restricciones constituye una necesidad (tal y como se señala también en [7]) y una importante linea futura de

investigación en el campo de los metaheuristicos para Programación Multiobjetivo.

Finalmente, a la vista de todo lo anterior podemos concluir que el campo de los metaheurísticos para la Programación multiobjetivo es un area de gran actividad actualmente, de una gran aplicabilidad y donde encontramos numerosas y prometedoras lineas futuras de investigación.

5 Bibliografía

[1] Abbass, H. A., The Self-Adaptive Pareto Differential Evolution Algorithm. In Congress on Evolutionary Computation (CEC2002), Vol. 1, páginas 831-836, Piscataway, New Jersey, May 2002.

[2] Alves, M.J., Climaco, J., An interactive method for 0-1 multiobjective problems using simulated annealing and tabú search. Journal of Heurisitcs, Vol. 6(3), páginas 385–403, 2000.

[3] Al-Yamani,A., Sait,S. Youssef,H, Barada H., Parallelizing Tabu Search on a Cluster of Heterogeneous Workstations. Journal of Heuristics, Vol. 8(3), páginas 277–304, 2002.

[4] P. Beausoleil, R.P., MOSS: multiobjective scatter search applied to nonlinear multiple criteria optimization, European Journal of Operational Research, Vol. 169 (2), páginas 426–449, 2006.

[5] Ben Abdelaziz, F., S. Krichen, Chaouachi, J., An Hybrid Metaheuristic for the Multiobjective Knapsack Problem. In Voss, S., Martello, S., Osman, I., Roucairol, C. (eds.) Metaheuristics-Advances and Trends in Local Search Paradigms for Optimization. Kluwer, páginas 205–212, 1999.

[6] Caballero, R., González, M., Guerrero, F.M., Molina, J. Paralera, C., Solving A Multiobjective Location Routing Problem With A Metaheuristic Procedure. Application To The Case Of Andalucia, European Journal Of Operational Research, to appear.

[7] Coello Coello, C.A. - Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art, Comput. Methods Appl. Mech. Engrg. 191, 1245-1287, 2002.

[8] Coello Coello, C.A., Van Veldhuizen, D.A., Lamont, G.B., Evolutionary Algorithms for Solving Multi-Objective Problems, Kluwer Academic Publishers, New York, 2002.

[9] Coello Coello, C.A., Cruz-Cortés, N., An Approach to Solve Multiobjective Optimization Problems Based on an Artificial Immune System, En Jonathan Timmis and Peter J. Bentley (editores), First International Conference on Artificial Immune Systems (ICARIS'2002), páginas 212–221, University of Kent at Canterbury, Inglaterra, 2002.

[10] Coello Coello, C.A., Lamont, G.B., Applications of multi-objective evolutionary algorithms, Advances in Natural Computation, Vol. 1 2005.

[11] Corne, D., Dorigo, M., Glover, F., New Ideas in Optimization. McGraw Hill, 1999.

[12] Corne, D., Knowles, J., Oates, M., The Pareto Envelope-based Selection Algorithm for Multiobjective Optimization. En Marc Shoenauer, Kalyanmoy Deb, Günter Rudolph, Xin Yao, Evelyne Lutton, J.J. Merelo y Hans-Paul Schwefel (eds.), Proceedings of the Parallel Problem Solving from Nature VI Conference, Springer. Lectures Notes in Computer Science Nº 1917, páginas 839–848, Paris, Francia, 2000.

[13] Czyzak P., Jaszkiewicz A., Pareto Simulated Annealing - A Metaheuristic technique for Multiple Objective Combinatorial Optimization. En Journal of Multicriteria Decision Analysis, Vol. 7, páginas 34–27, 1998.

[14] Dahl, G., Jörnsten, K., Lokketangen, A., A Tabú Search approach to the channel minimization problem. ICOTA'95, Chengdu, China, 1995.

[15] Das I., Dennis J., A closer look at drawbacks of minimizing weighted sums of objectives for Pareto set generation in multicriteria optimization problems En *Structural Optimization*, Vol. 14 (1), páginas 63–69, 1997.

[16] Deb, K., Agrawal, S., Pratab, A., Meyarivan, T., A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. KanGal report 200001, Indian Institute of Technology, Kanpur, India, 2000.

[17] Dorigo, M., Maniezzo, V., Colorni, A., The Ant System: Optimization by a Colony of Cooperating Agents. IEEE Transactions on Systems, Man, and Cybernetics-Part B, Vol. 26(1), páginas 29–41, 1996.

[18] Erhgott M., Gandibleux X., A survey and annotated bibliography on Multiobjective Combinatorial Optimization. OR Spektrum, Vol. 22, páginas 425–460, 2000.

[19] Ehrgott, M., Klamroth, K., Schwehm,C., A MCDM approach to portfolio optimization. Europen Journal of Operations Research, Vol. 155, páginas 752–770, 2003.

[20] Erickson, M., Mayer, A., Horn, J., The Niched Pareto Genetic Algorithm 2 Applied to the Design of Groundwater Remediation Systems. En Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele, Carlos Coello y David Corne (eds.) First International Conference on Evolutionary Multi-Criterion Optimization, páginas 681–695. Springer-Verlag. Lectures Notes in Computer Science Nº 1993, 2001.

[21] Feo, T., Resende, M., A probabilistic heuristic for a computationally difficult Set Covering Problem. Operations Research Letters. Vol. 8, páginas 67–71, 1989.

[22] Fonseca, C.M., Fleming, P.J., Genetic Algorithm for Multiobjective Optimization: Formulation, Discussion and Generalization. En S. Forest (ed.), Proceedings of Third International Conference on Genetic Algorithms. Morgan Kaufmann Publishers, páginas 416–423, 1993.

[23] Fourman, M.P., Compactation of symbolyc layout using genetic algorithms. En Genetic Algorithms and their Applications. Proceedings of the First International Conference on Genetic Algorithms and Their Applications. Lawrence Erlbaum, págian 141–153, 1985.

[24] Gandibleux, X., Mezdaoui, N., Freville, A., A tabú serach procedure to solve multiobjective combinatorial optimization problems. En Caballero, R., Ruiz, F. Steuer, R. (eds.) Advances in multiple objective and goal programming, Vol. 455. Lecture Notes in Economics and Mathematical Systems, páginas 291–300. Springer, Berlin.

[25] Gandibleux, X, Vancoppenolle, D., Tuyttens, D., A first making use of GRASP for solving MOCO problems. Technical Report, University of Valenciennes, France. Paper presented at MCDM 14, Charlottesville, VA, 1998.

[26] Gandibleux, X., Freville, A., Tabú search based procedure for solving the 0-1 multiobjective knapsack problem: the two objectives case. Journal of Heuristics, Vol. 6, páginas 361–383.

[27] Garey M., Johnson D., Computers and Intractability Freeman, 1979.

[28] Glover F., Parametric Combinations of Local Job Shop Rules, Chapter IV, ONR Research Memoramdum 117, Carnegie Mellon University, Pittsburgh, 1963.

[29] Glover F., Future Paths for Integer Programming and Links to Artificial Intelligence, En *Computers and Operations Research*, Vol. 13, páginas 533–549, 1986.

[30] Glover, F., Heuristics for Integer Programming Using Surrogate Constraints. Decision Sciencies, Vol. 8, páginas 156–166, 1977.

[31] Glover, F., Laguna, M., Tabú Search. Kluwer Academic Publishers. Boston, 1997.

[32] Glover, F., Tabu Search. En Gass, S. and Harris, C. (eds.) Encyclopedia of Operations Research and Management Science, 2nd edition. Kluwer. páginas 821–827, 2000.

[33] Goldberg, D.E., Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Publishing Co, 1989.

[34] Gravel, M., Price, W.L., Gagne, C., Scheduling continuous casting of aluminum using a multiple objective ant colony optimization metaheuristic. European Journal of Operational Research, Vol. 143(1), páginas 218–229, 2002.

[35] Guntsch, M., Middendorf, M., Solving Multi-Objective Permutation Problems with Population Based ACO, Proceedings of the Second International Conference on Evolutionary Multi-Criterion Optimization (EMO 2003), C.M. Fonseca, P.J. Fleming, E. Zitzler, K. Deb, L. Thiele (Eds.), Springer, LNCS 2636, páginas 464–478, 2003.

[36] Hansen, M.P., Tabú Search for Multiobjective Optimization: MOTS. 13th International Conference on Mutliple Criteria Decision Making. Cape Town, South Africa, 1997.

[37] Hernandez-Diaz, A., Santana, L., Coello, C., Caballero,R. Molina, J., A New Proposal For Multi-Objective Optimization Using Differential Evolution And Rough Sets Theory. 2006 Genetic And Evolutionary Computation Conference (Gecco'2006). Acm Press,Seattle, Washington, Usa. To appear.

[38] Hertz, A, Jaumard, B., Ribeiro, C., Formosinho, F.W., A multicriteria tabú search approach to cell formation problems in group technology with multiple objectives. Recherche Operationelle/Operations Research Vol. 28(3), páginas 303–328, 1994.

[39] Hapke M., Jaszkiewicz A., Slowinski R., Interactive analysis of Multicriteria Project Scheduling Problems. European Journal of Operational Research, Vol. 107, páginas 315–324, 1998.

[40] Hapke, M., Jaszkiewicz, A., Slowinski, R., Pareto Simulated Annealing for Fuzzy Multiobjective Combinatorial Optimization. Journal of Heuristics, 2000.

[41] Higgins, J., Hajkowicz, S., Bui, E., A multi-model for environmental investment decision making. Computers and Operations Research, to appear.

[42] Hilliard, M.R., Liepins, M. Palmer, M., Rangarajen, G., The computer as a partner in algorithmic design: Automated discovery of parameters for a multiobjective scheduling heuristic. En R. Sharda, B.L. Golden, E. Wasil, O. Balci and W. Stewart (eds.), Impact of Recent Computer Advances on Operations Research. North-Holland Publishing Company, 1989.

[43] Holland, J., Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, 1975.

[44] Horn, J., Nafpliotis, N., Multiobjective Optimization using the Niched Pareto Genetic Algorithm. Technical Report IlliGA1 Report 93005, Univiserty of Illinois at Urbana-Champaign, Illinois, USA, 1993.

[45] Jones, D.F., Mirrazavi, S.K., Tamiz, M., Multi-Objective Meta-heuristics: An overview of the currrent state of the art. European Journal of Operational Research, Vol. 137, páginas 1–9, 2002.

[46] Kennedy, J., Eberhart, R.C., Swarm Itelligence, Morgan Kaufmann Publishers, California, USA, 2001.

[47] Kirkpatrick, S., Gelatt, J.R., Vecchi, M.P., Optimization by Simulated Annealing. Science, Vol. 220, páginas 671–680, 1983.

[48] Knowles, J., Corne, D., Approximating the NonDominated Front using the Pareto Archived Evolution Strategy. Evolutionary Computation, Vol. 8(2), páginas 149–172, 2000.

[49] Laguna, M., Martí, R., Scatter Search. Kluwer Academic Publishers. Boston, 2003.

[50] Madavan, N. K., Multiobjective Optimization Using a Pareto Differential Evolution Approach. En CEC2002, Vol. 2, páginas 1145-1150, 2002.

[51] Molina, J., Laguna, M., Martí, R., Caballero, R., SSPMO: A Scatter Tabu Search Procedure for Non-Linear Multiobjective Optimization, INFORMS Journal of Computing, por aparecer, 2006.

[52] Moscato, P., On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Caltech Concurrent Computation Program, C3P Report 826, 1989.

[53] Nunes de Castro, L., Timmis, J., An Introduction to Artificial Immune Systems: A New Computational Intelligence Paradigm Springer-Verlag, 2002.

[54] Pareto V., Cours D'Economie Politique, Vol. I y II. F. Rouge, Lausanne, 1896.

[55] Parsopoulos, K., Taoulis, D., Pavlidis, N., Plagianakos, V., Vrahatis, M., Vector Evaluated Differential Evolution for Multiobjective Optimization. En CEC2004, Vol. 1, páginas 204-211, 2004.

[56] Ponce, M.T., Matos, M., Multicriteria distribution network planning using simulated annealing. International Transactions in Operational Research, Vol. 6(4), págians 377–391, 1999.

[57] Reynolds, R. G., An Introduction to Cultural Algorithms. En A. V. Sebald y L. J. Fogel (editores), Proceedings of the Third Annual Conference on Evolutionary Programming, páginas 131-139.World Scientific, River Edge, New Jersey, 1994.

[58] Ritzel, B.J., Eheart, W., Ranjithan, S., Using genetic algorithm to a solve a multiple objective groundwater pollution containment problem. Water Resources Research Vol. 30, páginas 1589–1603, 1994.

[59] Santana, L., Ramirez,N, Coello, C., Molina, J. Hernandez-Diaz, A., A New Proposal For Multiobjective Optimization Using Particle Swarm Optimization And Rough Sets Theory. Parallel Problem Solving From Nature (PPSN IX). Lecture Notes In Computer Sciences, Springer Verlag. To appear.

[60] Sareni, B., Krähenbühk, L., Fitness Sharing and Niching Methods Revisited. IEEE Transactions on Evolutionary Computation, Vol. 2 (3), páginas 97–106, 1998.

[61] Schaffer, J.D., Multiple Objective optimization with vector evaluated genetic algorithms. In Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms and Their Applications. Lawrence Erlbaum, páginas 93–100, 1985.

[62] Serafini, P., Simulated Annealing for multiobjective optimization problems. En Proceedings of the 10th International Conference on Multiple Criteria Decision Making. Taipei, Taiwan, Vol. 1, páginas 87–96, 1992.

[63] Simon H.A., Newell A., Heuristic Problem Solving: The Next Advance in Operations Research, En *Operations Research*, Vol. 6 (1), 1958.

[64] Srinivas, N., Deb, K., Multiobjective Optimization using Nondominated Sorting in Genetic Algorithm. Evolutionary Computation Vol. 2, páginas 221–248, 1994.

[65] Steuer R. E. - Multiple Criteria Optimization: Theory, Computation and Application,Wiley,New York,1986.

[66] Storn, R., Price, K., Differential Evolution - A Fast and Efficient Heuristic for Global Optimization over Continuous Spaces. Journal of Global Optimization, Vol. 11, páginas 341-359, 1997.

[67] Teghem, J., Tuyttens, D., Ulungu, E.L., An Interactive heuristic method for Multi-Objective Combinatorial Optimization. Computers & Operations Research. Vol. 27, páginas 621-634, 2000.

[68] Ulungu, E.L., Teghem, J., Fortemps, P., Heuristics for Multi-Objective Combinatorial Optimization Problems by Simulated Annealing. En Wei, Q., Gu, J., Chen, G., Wang, S. (Eds.) MCDM: Theory and Applications, páginas 228-238, 1995.

[69] Ulungu, E.L., Teghem, J., Ost, C., Efficiency of Interactive Multi-objective Simulated Annealing through a case study. Journal of the Operational Research Society, Vol. 49, páginas 1044-1050, 1998.

[70] Ulungu, E.L., Teghem, J., Fortemps, P., Tuyttens, D., MOSA method: A tool for solving Multi-Objective Combinatorial Optimization problems. Journal of Multi-Criteria Decision Analysis Vol. 8(4), páginas 221-236, 1999.

[71] Viana, A., Pinho de Sousa, J., Using metaheuristics in multiobjective resource constrained project scheduling. European Journal of Operational Research, Vol. 120 (2), págians 359-374, 2000.

[72] Vidal, R.V., Applied Simulated Annealing. Springer-Verlag. Berlin. 1993.

[73] Wilson, P.B., Macleod, M.D., Low implementations cost IIR digital filter design using genetic algorithms. In IEE/IEEE Workshop on Natural Algorithms in Signal Processing. Chelmsford, páginas 4/1-4/8, 1993.

[74] Zanakis S.H., Evans J.R., Heuristic Optimization: Why, When and How to use it, En *Interfaces*, Vol. 11 (5), 1981.

[75] Zanakis S.H., Evans, J.R., Vazacopoulus A.A., Heuristic Methods and Applications: a categorized survey, En *European Journal of Operational Research*, Vol. 43, páginas 88-110, 1989.

[76] Zitzler, E., Thiele, L., Multiobjective Evolutionary Algorithms: A comparative case study and the strength pareto approach. IEEE Transactions on Evolutionary Computation, Vol. 3(4), páginas 257-271, 1999.

[77] Zitzler, E., Laumanns, M., Thiele, L., SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, 2001.

Una Visión General de los Algoritmos Meméticos*

Carlos Cotta
Dept. Lenguajes y Ciencias de la Computación, ETSI Informática,
University of Málaga, Campus de Teatinos, 29071 - Málaga, Spain.

ccottap@lcc.uma.es

1 Introducción

Uno de los periodos clave en la historia de la Optimización como disciplina lo constituyen sin duda alguna las primeras décadas de la segunda mitad del siglo XX. En un tiempo en el que la concepción natural de la resolución de problemas era la obtención de la solución óptima al mismo (o cuanto menos de una solución con una garantía de aproximación al óptimo), empezó a tomar cuerpo una desagradable realidad: existían muchos problemas de utilidad cierta para los que no sólo resultaba insostenible plantear una resolución exacta, sino que ni tan siquiera un enfoque aproximado con garantías realistas era aceptable en la práctica. Esto abrió el camino a diferentes líneas de investigación para dar respuesta a esta problemática, y que más adelante desembocarían en lo que hoy se conoce como *metaheurísticas*. Entre las mismas, deben destacarse los algoritmos evolutivos [1, 2, 3, 4] (EAs[1]) por estar íntimamente relacionados con el tema que nos ocupa: los algoritmos meméticos.

A pesar de que estas técnicas fueran consideradas en su momento por parte de la comunidad científica como un "reconocimiento de la derrota", el tiempo ha demostrado su utilidad como punta de lanza tecnológica en la optimización de problemas reales. Por supuesto, este éxito no es exclusivo de los EAs, sino que

*El autor agradece el apoyo parcial del MCyT a través del contrato TIN2005-08818-C04-01.

[1]En este y en sucesivos acrónimos se empleará la versión inglesa por motivos de consistencia con la literatura, y para evitar posibles fuentes de confusión o ambigüedad.

se extiende a otras técnicas relacionas tales como el recocido simulado [5] (SA), la búsqueda tabú [6] (TS), etc. Parte de la justificación (y de hecho, del enfoque metodológico) de los algoritmos meméticos (MAs) se haya precisamente en el éxito de métodos de optimización tan diversos. En este sentido, debe considerarse que en ciertos campos se desarrolló una tendencia al purismo algorítmico, esto es, a no apreciar como característica esencial de estas técnicas su flexibilidad y capacidad de asimilación de elementos algorítmicos externos o ad hoc, que pudieran acercar la técnica de resolución al problema que resolver en cada momento. No fue hasta mediados de los noventa cuando la formulación del así denominado Teorema de *No Free Lunch* por Wolpert and Macready [7] dio pie a una suerte de catarsis, a partir de la cual quedó definitivamente claro que un algoritmo de búsqueda u optimización se comporta en estricta concordancia con la cantidad y calidad del conocimiento específico del problema que incorpora. Mirando retrospectivamente a estos años nos encontramos con que esta filosofía que comenzó a imponerse de manera generalizada a finales del siglo XX ya estaba siendo promulgada de hecho con anterioridad por diversos investigadores, e.g., Hart and Belew [8], Davis [9], y Moscato [10]. El paradigma de los MAs surgiría precisamente a partir del trabajo de Pablo Moscato [11, 12, 13].

Los MAs son una familia de mataheurísticas que intentan aunar ideas y conceptos de diferentes técnicas de resolución, como por ejemplo EAs y TS. El adjetivo "memético" viene del término inglés *meme*, acuñado por R. Dawkins [14] para designar al análogo del gen en el contexto de la evolución cultural. Resulta conveniente resaltar sin embargo que el empleo de esta terminología no representa un propósito de adherirse a una metáfora de funcionamiento concreta (la evolución cultural en este caso), sino más bien lo contrario: hacer explícito que se difumina la inspiración puramente biológica, y se opta por modelos más genéricos en los que se manipula, se aprende y se transmite información. En relación con esto último y a la forma en la que más comúnmente un MA puede implementarse, pueden encontrarse diversos trabajos que hacen uso de nombre alternativos para referirse a éstos (e.g., EAs híbridos o lamarckianos), o que aun usando el propio término MA, hacen una interpretación muy restringida del mismo. Sea como fuere, puede decirse que un MA es una estrategia de búsqueda en la que una población de agentes optimizadores compiten y cooperan de manera sinérgica [10]. Más aún, estos agentes hacen uso explícito de conocimiento sobre el problema que se pretende resolver, tal como sugiere tanto la teoría como la práctica [15]. La siguiente sección proporciona una descripción algorítmica más detallada de los MAs.

2 Un Algoritmo Memético Básico

Los MAs son metaheurísticas basadas en población. Esto quiere decir que mantienen un conjunto de soluciones candidatas para el problema considerado.

De acuerdo con la jerga empleada en EAs, cada una de estas soluciones tentativas es denominada un *individuo*. Tal como se anticipó anteriormente, la naturaleza de los MAs sugiere que el término *agente* es no obstante más apropiado. El motivo básico es el hecho de que "individuo" denota un ente *pasivo* que está sujeto a los procesos y reglas evolutivas, mientras que el término "agente" implica la existencia de un comportamiento *activo*, dirigido a propósito a la resolución de un cierto problema. Dicho comportamiento activo se ve reflejado en diferentes constituyentes típicos del algoritmo, como por ejemplo técnicas de búsqueda local.

La Figura 1 muestra el esquema general de un MA. Como en los EAs, la población de agentes está sujeta a los procesos de competición y cooperación mutua. Lo primero se consigue a través de los bien conocidos procedimientos de selección (línea 6) y reemplazo (línea 12): a partir de la información que proporciona una función de guía ad hoc se determina la bondad de los agentes en *pop*; acto seguido, se selecciona una parte de los mismos para pasar a la fase reproductiva atendiendo a dicha bondad. Posteriormente, se vuelve a hacer uso de esta información para determinar qué agentes serán eliminados de la población para hacer sitio a los nuevos agentes. En ambos casos –selección y reemplazo– pueden usarse cualesquiera de las estrategias típicas de los EAs, e.g., torneo, ranking, elitismo, etc.

En cuanto a la cooperación, ésta se consigue a través de la reproducción. En esta fase se crean nuevos agentes a partir de los existentes mediante el empleo de una serie de operadores de reproducción. Tal como se muestra en la Figura 1, líneas 7–11, pueden considerarse un número arbitrario $\#op$ de tales operadores, que se aplican secuencialmente a la población de manera segmentada, dando lugar a varias poblaciones intermedias $auxpop[i]$, $0 \leqslant i \leqslant \#op$, donde $auxpop[0]$ está inicializada a *pop*, y $auxpop[\#op]$ es la descendencia final. En la práctica, la situación más típica es la de utilizar simplemente tres operadores: recombinación, mutación, y mejora local. Apréciese en la línea 9 del pseudocódigo que estos operadores reciben no sólo las soluciones sobre las que actúan, sino también la instancia I que se desea resolver. Con esto se ilustra el hecho de que los operadores de un MA son conscientes del problema, y basan su funcionamiento en el conocimiento que incorporan sobre el mismo (a diferencia de los modelos más clásicos de EA).

Uno de los procesos reproductivos que mejor encapsula la cooperación entre agentes (dos, o más [16]) es la recombinación. Esto se consigue mediante la construcción de nuevas soluciones a partir de la información relevante contenida en los agentes cooperantes. Por "relevante" se entiende que los elementos de información considerados tienen importancia a la hora de determinar (en un sentido o en otro) la calidad de las soluciones. Ésta es sin duda una noción interesante que se aleja de las más clásicas manipulaciones sintácticas, típicas de EAs simples. Volveremos a esto más adelante, en la próxima sección.

El otro operador clásico –la mutación– cumple el rol de "mantener vivo el

Algoritmo Memetico

ENTRADA: una instancia I de un problema P.
SALIDA: una solucion *sol*.

```
// generar poblacion inicial
para j ← 1:popsize hacer
  sea ind ← GenerarSolucionHeuristica (I)
  sea pop[j] ← MejoraLocal (ind, I)
finpara
repetir // bucle generacional
  // Seleccion
  sea criadores ← SeleccionarDePoblacion (pop)
  // Reproduccion segmentada
  sea auxpop[0] ← pop
  para j ← 1:#op hacer
    sea auxpop[j] ← AplicarOperador (op[j], auxpop[j − 1], I)
  finpara
  sea newpop ← auxpop[#op]
  // Reemplazo
  sea pop ← ActualizarProblacion (pop, newpop)
  // Comprobar convergencia
  si Convergencia (pop) entonces
    sea pop ← RefrescarPoblacion (pop, I)
  finsi
hasta CriterioTerminacion (pop, I)
devolver Mejor (pop, I)
```

Figura 1: Plantilla general de un MA

fuego", inyectando nueva información en la población de manera continua (pero a ritmo bajo, ya que de lo contrario el algoritmo se degradaría a una pura búsqueda aleatoria). Por supuesto, esta interpretación es la que proviene del área de los algoritmos genéticos [17], y no necesariamente coincide con la otros investigadores (aquellos del área de la programación evolutiva [1] sin ir más lejos). De hecho, en ocasiones se ha aducido que la recombinación no es es más que una *macro-mutation*, y ciertamente ese puede ser el caso en numerosas aplicaciones de los EAs en los que este operador de recombinación simplemente realiza una mezcla aleatoria de información. Sin embargo, no cabe hacer una apreciación similar en el campo de los MAs, ya que en éstos la recombinación se realiza típicamente mediante el empleo de estrategias *astutas*, y por lo tanto contribuyen de manera esencial a la búsqueda.

Finalmente, una de las características más distintivas de los MAs es el empleo de estrategias de búsqueda local (LS). Éstas (nótese que pueden emplearse diferentes estrategias de LS en diferentes puntos del algoritmo) constituyen una de las razones esenciales por las que es apropiado usar el término "agente" en este contexto: su funcionamiento es local, y en ocasiones incluso autónomo. De esta manera, un MA puede verse como una colección de agentes que realizan una exploración autónoma del espacio de búsqueda, cooperando en ocasiones a través de la recombinación, y compitiendo por recursos computacionales a través de los mecanismos de selección/reemplazo.

El pseudocódigo de la Figura 1 muestra un componente que merece asimismo atención: el procedimiento RefrescarPoblación (líneas 13–15). Este procedimiento tiene suma importancia con vistas al aprovechamiento de los recursos computacionales: si en un determinado instante de la ejecución todos los agentes tienen un estado similar (esto es, se ha producido convergencia), el avance de la búsqueda se torna muy complejo. Este tipo de circunstancias puede detectarse a través del empleo de medidas tales como la entropía de Shannon [18], fijando un umbral mínimo por debajo del cual se considera que la población ha degenerado. Obviamente, dicho umbral depende de la representación de problema que se esté usando, y debe decidirse por lo tanto de manera particular en cada caso.

3 Diseño de MAs Efectivos

Atacar un cierto problema de optimización con MAs requiere instanciar la plantilla genérica descrita anteriormente, empleando para ello conocimiento del problema. Dado que el diseño de un algoritmo de búsqueda efectivo es en general tan complejo como los propios problemas que se desean resolver, nos encontramos ante la tesitura de tener que emplear directrices heurísticas para abordar dicho problema de diseño. A continuación se consideraran algunas de estas directrices para algunos de los componentes esenciales de los MAs.

3.1 Representación

El primer elemento que debe determinarse es la representación de las soluciones que se va a usar. Es importante en este punto aclarar que *representación* no debe entenderse como meramente *codificación*, algo para lo que lo relevante son consideraciones relativas a consumo de memoria, complejidad de manipulación, etc. Muy al contrario, la representación hace referencia a la formulación abstracta de las soluciones desde el punto de vista del algoritmo [19]. En este sentido, recuérdese la mención a *información relevante* que se hizo en la Seccción 2. Dada una cierta representación de las soluciones, éstas pueden entenderse como compuestas de determinadas *unidades de información*; si los operadores que emplea el MA son conscientes del problema atacado, las unidades de información que

identifiquen deben servir para determinar si una solución es buena/prometedora o no. La dinámica del sistema debe entonces tender a retener las unidades de información que lleven asociadas un efecto positivo, y a eliminar aquellas que tengan connotación negativa.

El siguiente ejemplo puede ayudar a ilustrar este aspecto de la representación. Considérese un problema definido sobre un espacio de soluciones compuesto de todas las permutaciones de n elementos; estas soluciones pueden entenderse como compuestas por diferentes tipos de información [20], e.g.,

- *posicional*, i.e., el elemento e aparece en la posición j.
- *precedencia*, i.e., el elemento e aparece antes/después que el elemento e'.
- *adyacencia*, i.e., el elemento e aparece junto al elemento e'.

La relevancia de cada tipo de información dependerá obviamente del problema que se desea resolver. Por ejemplo, la información de adyacencia es importante para el problema del viajante de comercio (TSP), pero no así la información posicional. Por otra parte, se ha comprobado que esta última sí es relevante en problemas de planificación de cadenas de montaje (*flowshop scheduling*) (FS) [21], siendo la información de adyacencia menos importante en este caso. Esto quiere decir que un operador de recombinación como ER (*edge-recombination*) [22] funcionará mejor que un operador basado en información posicional como PMX (*partially-mapped crossover*) [23] o UCX (*uniform cycle crossover*) [21] en el TSP, pero los últimos funcionarán mejor sobre FS.

No es sorprendete a la vista de lo anterior que la obtención de métodos o medidas para cuantificar la bondad de una cierta representación para un cierto problema haya sido y sea un tema de gran interés. Ha habido diferentes propuestas en este sentido: *epistasis* (i.e., la influencia no aditiva que sobre la función objetivo tiene la combinación de varias unidades de información) [24, 25], *varianza en la adecuación de formas* (i.e., varianza en los valores que devuelve la función objetivo para soluciones que comparten un cierto conjunto de unidades de información) [26], y *correlación de adecuación* (correlación entre los valores de la función objetivo para entre unas soluciones y sus descendientes directos) [27, 28]. Debe reseñarse que además de usar una métrica para predecir cuán bueno puede ser el rendimiento de un cierto operador pre-existente (i.e., *análisis inverso*), pueden definirse nuevos operadores ad hoc para manipular la mejor representación (*análisis directo*) [13].

Sea cual fuere la métrica usada para cuantificar la bondad de una representación concreta, hay otras consideraciones que también pueden jugar un papel determinante en el rendimiento final del algoritmo, tales como por ejemplo la existencia de restricciones en el espacio de búsqueda. Esta última problemática puede atacarse de tres maneras: (i) usando funciones de penalización que dirijan la búsqueda hacia regiones factibles, (ii) usando mecanismos de reparación que produzcan soluciones factibles a partir de soluciones infactibles, y (iii) usando

operadores reproductivos que permanezcan siempre dentro de la zona factible. En los dos primeros casos es posible mantener la complejidad de la representación a un nivel más bajo (aunque lógicamente el algoritmo se beneficiará de cualquier conocimiento que adicional que pudiera usarse aquí). En el tercer caso, es responsabilidad de la representación o de los operadores el garantizar la factibilidad, y esto conllevará una complejidad adicional. Así, es posible definir representaciones indirectas que mediante el empleo de *decodificadores* garanticen la factibilidad de las soluciones representadas. La idea básica es utilizar un mecanismo sofisticado para pasar del genotipo al fenotipo, de manera que no sólo se consigan soluciones factibles, sino que además se introduzca conocimiento del problema que facilite que éstas sean de calidad (e.g., [29, 30, 31] entre otras).

3.2 Operadores Reproductivos

La generación de nuevas soluciones durante la fase reproductiva se realiza mediante la manipulación de las unidades de información relevantes que se han identificado. A tal fin, puede emplearse cualquiera de las plantillas genéricas definidas para ello, e.g., RRR (*random respectful recombination*), RAR (*random assorting recombination*), y RTR (*random transmitting recombination*) entre otras [32]. En cualquier caso, huelga decir que el rendimiento de algoritmo se verá beneficiado si en lugar de manipular las unidades de información a ciegas, se hace de manera inteligente empleando conocimiento del problema. Desde un punto de vista general, esta inclusión de conocimiento del problema en la manipulación de las unidades de información tiene dos vertientes: la selección de las características parentales que serán transmitidas a la descendencia, y la selección de las características no-parentales que serán incluidas en la misma.

En relación a la selección de la información contenida en los padres que debe transmitirse a los hijos, la evidencia experimental aconseja conservar aquellas características comunes a ambos padres (e.g., [22, 33]). Una vez hecho esto, el descendiente puede completarse de diferentes maneras. Así, Radcliffe y Surry [26] proponen el empleo de estrategias de búsqueda local o de esquemas de enumeración implícitos. Estos últimos pueden usarse también para encontrar la mejor combinación posible de la información parental [34, 35, 36, 37] (dependiendo de las características de la representación, sería posible que esta combinación no necesariamente respetara las propiedades comunes). Puede apreciarse fácilmente que este tipo de recombinación sería monótono en el sentido de que los hijos serían siempre al menos tan buenos como los padres.

Hasta cierto punto podría hacerse un análisis similar del operador de mutación, si bien es verdad que éste juega un papel bien distinto: introducir nueva información en la población. En principio, esto puede conseguirse mediante la eliminación de ciertas unidades de información de una solución, y su substitución por información puramente aleatoria, o por información obtenida por alguno de

los métodos de completado descritos anteriormente. Sin embargo, hay que resaltar que el papel de la mutación tiene ciertos matices diferenciadores en MAs frente a los clásicos EAs. De hecho, es posible incluso que un MA no tenga un operador de mutación diferenciado, sino que éste esté simplemente empotrado en la búsqueda local, e.g., véase [38, 39]. Uno de los motivos es el hecho de que los MAs están dotados de mecanismos de reinicio de la población (véase la Sección 3.4), y en ciertos contextos puede ser mejor dejar converger la población rápidamente y luego reiniciar, que diversificar constantemente la búsqueda. En cualquier caso, hay situaciones en las que mutación sí adquiere un papel determinante, y en las que incluso se emplean varios operadores de mutación. Esto se realiza bien por el empleo de diferentes vecindades (e.g., [40]), o definiendo mutaciones débiles y fuertes que introduzcan diferentes niveles de perturbación (e.g., [41]). Nótese que en cierto sentido el empleo de diferentes operadores reproductivos implica de manera implícita la consideración de diferentes representaciones y/o vecindades durante la ejecución, muy en la línea de los que se hace en la búsqueda en vecindades variables [42] (VNS).

Es posible introducir también conocimiento del problema mediante el empleo de heurísticas constructivas en los operadores de inicialización usados para la generación de la población inicial (Figura 1, línea 2). Por ejemplo, se han empleado estrategias voraces para este propósito en [43, 44].

3.3 Búsqueda Local

La presencia de componentes de búsqueda local (LS) es –tal como se comento anteriormente– una de las características más distintivas de los MAs. El hecho de que la mayoría de los MAs incorporen LS es una de las causas por las que a veces se pueden encontrar simplificaciones del tipo MA = EA + LS, y que deben evitarse; véase [11, 12, 13] para más detalles. De hecho, es posible encontrar enfoques metaheurísticos con muy similar filosofía a la de los MAs, y que sin embargo no pueden llamarse evolutivos a no ser que se asuma una definición tan amplia del termino que prácticamente abarque a cualquier método basado en población. La técnica de búsqueda dispersa (SS) [45] es un buen ejemplo en este sentido. Por otra parte, no es extraño encontrar enfoques evolutivos en los que el conocimiento del problema se concentra más en el operador de recombinación que en el uso de una búsqueda local, e.g., [35, 46]. En cualquier caso, esta claro que EA + LS $\subset$ MA, y que el componente LS es típicamente uno de los que más contribuyen al éxito del algoritmo.

Las técnicas de mejorar local pueden modelarse como trayectorias en el espacio de búsqueda tal que soluciones vecinas en dicha trayectoria difieren en una pequeña cantidad de unidades de información. Esta definición idealizada puede requerir no obstante diferentes matizaciones si por ejemplo se emplea TS para este fin, e.g., [47, 36, 48] entre otras muchas. Así, es normal que muchas imple-

mentaciones de TS usen estrategias de intensificación que hagan que en ciertos momentos la búsqueda se continúe por ciertas soluciones anteriores de calidad (así, más que a un camino lineal, el recorrido a través del espacio de búsqueda se asemejaría a una trayectoria ramificada). Más aún tanto en TS como en otras metaheurísticas tales como SA, puede darse que la calidad de las soluciones no se incremente de manera monótona, sino que en ciertos momentos empeore con la finalidad de poder escapar de óptimos locales. Por supuesto, al final de la ejecución del procedimiento se conserva la mejor solución encontrada, y no la última generada.

A la hora de implementar el componente LS es importante determinar el criterio de terminación. Si se está empleando una técnica simple de escalada (HC) puede tener sentido determinar si la solución actual es un óptimo local y detener el procedimiento únicamente en ese caso. Obviamente, esto no es posible si se usa TS o SA, ya que estas técnicas tienen capacidades globales de optimización, por lo que lo más común es definir un tope computacional máximo (e.g., en forma de número de soluciones exploradas). Lógicamente, en este caso la solución final no tiene por qué ser un óptimo local (como algunas descripciones erróneas de MAs aseguran). Además, debe encontrarse un equilibrio adecuado entre el esfuerzo computacional que se realiza durante LS y el que la búsqueda poblacional subyacente realiza. La importancia de este hecho ha dado lugar a la noción de *lamarckismo parcial* [49, 50], esto es, no usar siempre la búsqueda local, sino únicamente sobre algunas soluciones o bien seleccionadas aleatoriamente, o bien en función de su calidad, o bien según algún otro método (véase también [51]).

Del mismo modo que se pueden definir métricas para cuantificar la bondad de una representación (u operador que trabaje sobre la misma), pueden definirse métricas que ayuden a predecir si una determinada definición de vecindad puede ser beneficiosa. Por ejemplo, la correlación entre *distancia* y adecuación [52, 53] (FDC) es una de las propuestas. Esencialmente, la distancia mencionada se entiende como el número de movimientos (saltos de vecindad) que hay que realizar para pasar de un óptimo local al óptimo global. Si el coeficiente de correlación entre esta distancia y la calidad de la función objetivo es alta, entonces la calidad de las soluciones tiende a mejorar al acercarse al óptimo global, y la dinámica evolutiva del MA lo llevará a su cercanía. Si la correlación fuera negativa, el problema sería engañoso para el MA, ya que los óptimos locales mejores se alejarían del óptimo global.

Otro aspecto importante en relación al paisaje de búsqueda es su topología global, y más precisamente si la relación de vecindad es regular o no, y que relación guarda con la calidad de las soluciones. Bierwirth *et al.* [54] han estudiado esta circunstancia para un problema de planificación, y han encontrado que las mejores soluciones tienen una mayor conectividad, lo que las hace más fácilmente alcanzables, incluso por mor de pura deriva genética. No todos los problemas tienen esta propiedad, y de hecho, Cotta y Fernández han encontrado que la

representación directa para la búsqueda de reglas de Golomb de tamaño mínimo tiene precisamente la propiedad opuesta [55].

3.4 Gestión de la Diversidad

Hay diferentes maneras de enfocar la diversidad en algoritmos basados en población. Por un lado pueden considerarse métodos de preservación de la diversidad, dentro de los cuales se engloban claramente los operadores de mutación. Éstos no son los únicos mecanismos posibles sin embargo. Por ejemplo, pueden introducirse soluciones completamente nuevas (los así llamados "inmigrantes aleatorios") [56] en la población, e.g., [57], o pueden emplearse poblaciones con estructura espacial [58]. En este último caso, se restringe el emparejamiento de agentes o el reemplazo de los mismos a elementos situados en posiciones vecinas dentro de la estructura topológica de la población. Esto causa un ralentizamiento de la propagación de información a través de la población, con lo que se impide (o al menos dificulta) que algunos *super-agentes* tomen rápidamente control de la misma y destruyan toda diversidad.

En la literatura se han propuesto diferentes topologías para organizar la población, i.e., anillos, rejillas, hipercubos, etc. En relación con los MAs, una de las opciones más exitosas has sido una estructura jerárquica en forma de árbol ternario [59, 41, 60, 61]. Esta topología se ha combinado con una estrategia para organizar la distribución de las soluciones en función de su calidad. Más concretamente, cada nodo del árbol está restringido a tener una solución mejor que cualquiera de los nodos descendientes. Esto implica que cuando un agente tiene una solución mejor que la de su antecesor directo en el árbol, las intercambian. De esta manera, hay un continuo flujo de soluciones de calidad hacia la parte superior del árbol, lo cual también garantiza que cuando se realiza una recombinación, las soluciones que toman parte en ella son de calidad similar.

Como complemento a los mecanismos de preservación anteriores se pueden considerar también los mecanismos de restauración de la diversidad: cuando se detecta que la diversidad ha caído por debajo de un cierto umbral, o cuando la dinámica del algoritmo apunta a un estado de degeneración en la búsqueda [62] se activa uno de estos mecanismos para relanzarla. Una posibilidad en este sentido es emplear hipermutación [63, 41] (cf. mutación pesada, véase Sección 3.2). Alternativamente, la población puede refrescarse mediante la llegada masiva de inmigrantes aleatorios que sustituyan a toda la población, salvo a algunas soluciones de elite.

4 Aplicaciones de los MA

Uno de los campos más fructíferos para los MAs es el ámbito de la optimización combinatoria, para el que estas técnicas cuentan con cientos de aplicaciones. Eso

no es sorprendente si tenemos en consideración que existen miles de problemas de optimización pertenecientes a la clase NP, donde los MA se han mostrado de gran valor. De entre todas éstas, y a modo ilustrativo, pueden destacarse las siguientes: problemas de particionado en grafos [64, 65], partición de números [66, 59], conjunto independiente de cardinalidad máxima [67, 68], empaquetado [69], coloreado de grafos [70, 71], recubrimiento de conjuntos [72], planificación de tareas en una máquina con tiempos de "set-up" y fechas de entrega [73, 74], planificación de tareas en varias máquinas [75, 76], problemas de asignación generalizados [77], problemas de mochila multidimensional [78, 79], programación entera no-lineal [80], asignación cuadrática [81, 53], particionado de conjuntos [82], y muy especialmente el *problema del viajante de comercio* [83, 53, 84]. Es de destacar que en una gran parte de estas publicaciones los propios autores destacan que la metodología constituye el estado del arte para el problema en consideración, lo que es de interés debido a que estos son problemas "clásicos" en el área de la optimización combinatoria.

El paradigma fue utilizado en otros problemas menos conocidos, pero sin duda de igual importancia, como son: emparejamiento parcial de formas geométricas [85], optimización en "paisajes NK" [86], diseño de trayectorias óptimas para naves espaciales [87], asignación de frecuencias [88], construcción de árboles de expansión mínimos con restricciones de grado [89], problemas de emplazamiento [90, 91], optimización de rutas [92], problemas de transporte [93, 94], isomorfismos en grafos [95], problemas de biconexión de vértices [96], agrupamiento [97], telecomunicaciones [98], búsqueda de regleros de Golomb mínimos [99, 100], búsqueda de patrones estables en autómatas celulares [36, 101], identificación de sistemas no-lineares [102], programación de tareas de mantenimiento [103, 104], *open shop scheduling* [105, 40], *flowshop scheduling* [106, 44], planificación de proyectos [107, 108], planificación de almacén [109], planificación de producción [110, 111], confección de horarios [112, 113], planificación de turnos [114, 115], planificación de juegos deportivos [116] y planificación de exámenes [117, 118].

Los MAs, también han sido citados en la literatura de aprendizaje en máquinas y robótica como algoritmos genéticos híbridos. Destacamos algunas aplicaciones como por ejemplo: entrenamiento de redes neuronales [119, 120], reconocimiento de características [121], clasificación de características [122, 123], análisis de series temporales [124], aprendizaje de comportamientos reactivos en agentes móviles [125], planificación de trayectorias [126, 127], control óptimo [128], etc.

En las áreas de la Electrónica y la Ingeniería podemos destacar: proyectos de VLSI [129], optimización de estructuras [130] y mecánica de fracturas [131], modelado de sistemas [132], control de reactores químicos [133], calibración de motores [134], problemas de diseño óptimo en Aeronáutica [135, 136], diseño de sistemas ópticos [137], control de tráfico [138], y planificación en problemas de potencia [139] entre otros.

Otras aplicaciones de estas técnicas pueden encontrarse en: Medicina [140,

141], Economía [142, 143], Oceanografia [144], Matemáticas [145, 146, 147], Procesamiento de imágenes y de voz [148, 149, 150], y un sinnúmero de ellas en Bioinformática ([49, 60, 151, 152] entre otras muchas).

5 Conclusiones

A diferencia de otras técnicas de optimización, los MAs fueron explícitamente concebidos como un paradigma ecléctico y pragmático, abierto a la integración de otras técnicas (metaheurísticas o no). En última instancia, esta habilidad para combinar de manera sinergética diferentes métodos es una de las razones de su éxito. Los MAs proporcionan un marco de trabajo apropiado para integrar en un único motor de búsqueda diferentes heurísticas provechosas. En este sentido, los MAs deben considerarse no como competidores, sino como integradores: allá donde una metaheurística *pura* empiece a alcanzar sus límites, los MAs constituyen el siguiente paso natural.

Aunque existe un importante componente experimental en el diseño de los MAs, no por ello puede afirmarse que el paradigma se reduce a combinar varias técnicas y realizar pruebas experimentales para comprobar si es satisfactoria. Muy al contrario, todo el corpus teórico disponible tanto para técnicas basadas en población como para técnicas de búsqueda local es de aplicación en el diseño de un MA. Otras estrategias de gran interés en este área son el diseño por analogía, y el máximo aprovechamiento de los recursos computaciones. Téngase en cuenta en relación a esto último que una técnica de búsqueda local muy sofisticada puede proporcionar mejores resultados que un simple HC, pero necesitar mucho más tiempo de cómputo para ello. En problemas en los que el coste de evaluar una solución es grande, o en los que los tamaños de las vecindades son considerables, éste es un problema que debe tenerse muy en cuenta.

Está claro asimismo que nuestro mundo se está haciendo cada vez más complejo a un ritmo acelerado, al menos desde un punto de vista tecnológico. Los años venideros depararán nuevos desafíos desde el punto de vista de la optimización a los que habrá que dar respuesta con metaheurísticas. No sólo habrá que hacer frente a problemas de optimización a gran escala, sino que estos mismos serán cada vez más complejos per se. Para ello, las técnicas de optimización tendrán que adaptarse a esta complejidad, dejando de lado los tradicionales enfoques unidimensionales y puramente secuenciales. Así, algunos de los aspectos de los MAs que tomarán cada vez más relevancia son la optimización multi-objetivo [51, 153, 154], la auto-adaptación [155, 156], y el funcionamiento autónomo [157, 158]. Como puede apreciarse, ya hay algunas propuestas en este sentido, siendo posible además aprovechar ideas de técnicas relacionadas tales como las hiperheurísticas [159, 160]. Otros métodos jugarán también un papel esencial, e.g., las técnicas de reducción a un kernel seguro comúnmente empleadas dentro del campo de la complejidad parametrizada [161]. Dado que el eclecticismo

es esencial para adaptarse a este nuevo escenario, sólo cabe decir que el futuro es prometedor para los MAs.

6 Bibliografía

[1] Fogel, L.J., Owens, A.J., Walsh, M.J.: Artificial Intelligence through Simulated Evolution. John Wiley & Sons, New York (1966)

[2] Holland, J.: Adaptation in Natural and Artificial Systems. University of Michigan Press (1975)

[3] Rechenberg, I.: Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog, Stuttgart (1973)

[4] Schwefel, H.P.: Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik. Diplomarbeit, Technische Universität Berlin, Hermann Föttinger–Institut für Strömungstechnik (1965)

[5] Kirkpatrick, S., Gelatt Jr., C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220** (1983) 671–680

[6] Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers, Boston, MA (1997)

[7] Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation **1(1)** (1997) 67–82

[8] Hart, W.E., Belew, R.K.: Optimizing an arbitrary function is hard for the genetic algorithm. In Belew, R.K., Booker, L.B., eds.: Proceedings of the 4^{th} International Conference on Genetic Algorithms, San Mateo CA, Morgan Kaufmann (1991) 190–195

[9] Davis, L.D.: Handbook of Genetic Algorithms. Van Nostrand Reinhold Computer Library, New York (1991)

[10] Moscato, P.: On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Technical Report Caltech Concurrent Computation Program, Report. 826, California Institute of Technology, Pasadena, California, USA (1989)

[11] Moscato, P.: Memetic algorithms: A short introduction. In Corne, D., Dorigo, M., Glover, F., eds.: New Ideas in Optimization. McGraw-Hill, Maidenhead, Berkshire, England, UK (1999) 219–234

[12] Moscato, P., Cotta, C.: A gentle introduction to memetic algorithms. In Glover, F., Kochenberger, G., eds.: Handbook of Metaheuristics. Kluwer Academic Publishers, Boston MA (2003) 105–144

[13] Moscato, P., Cotta, C., Mendes, A.S.: Memetic algorithms. In Onwubolu, G.C., Babu, B.V., eds.: New Optimization Techniques in Engineering. Springer-Verlag, Berlin Heidelberg (2004) 53–85

[14] Dawkins, R.: The Selfish Gene. Clarendon Press, Oxford (1976)

[15] Culberson, J.: On the futility of blind search: An algorithmic view of "No Free Lunch". Evolutionary Computation **6** (1998) 109–127

[16] Eiben, A.E., Raue, P.E., Ruttkay, Z.: Genetic algorithms with multi-parent recombination. In Davidor, Y., Schwefel, H.P., Männer, R., eds.: Parallel Problem Solving From Nature III. Volume 866 of Lecture Notes in Computer Science. Springer-Verlag (1994) 78–87

[17] Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading, MA (1989)

[18] Davidor, Y., Ben-Kiki, O.: The interplay among the genetic algorithm operators: Information theory tools used in a holistic way. In Männer, R., Manderick, B., eds.: Parallel Problem Solving From Nature II, Amsterdam, Elsevier Science Publishers B.V. (1992) 75–84

[19] Radcliffe, N.J.: Non-linear genetic representations. In Männer, R., Manderick, B., eds.: Parallel Problem Solving From Nature II, Amsterdam, Elsevier Science Publishers B.V. (1992) 259–268

[20] Fox, B.R., McMahon, M.B.: Genetic operators for sequencing problems. In Rawlins, G.J.E., ed.: Foundations of Genetic Algorithms I, San Mateo, CA, Morgan Kaufmann (1991) 284–300

[21] Cotta, C., Troya, J.M.: Genetic forma recombination in permutation flowshop problems. Evolutionary Computation **6** (1998) 25–44

[22] Mathias, K., Whitley, L.D.: Genetic operators, the fitness landscape and the traveling salesman problem. In Männer, R., Manderick, B., eds.: Parallel Problem Solving From Nature II, Amsterdam, Elsevier Science Publishers B.V. (1992) 221–230

[23] Goldberg, D.E., Lingle Jr., R.: Alleles, loci and the traveling salesman problem. In Grefenstette, J.J., ed.: Proceedings of the 1^{st} International Conference on Genetic Algorithms, Hillsdale NJ, Lawrence Erlbaum Associates (1985) 154–159

[24] Davidor, Y.: Epistasis Variance: Suitability of a Representation to Genetic Algorithms. Complex Systems **4** (1990) 369–383

[25] Davidor, Y.: Epistasis variance: A viewpoint on GA-hardness. In Rawlins, G.J.E., ed.: Foundations of Genetic Algorithms I, San Mateo, CA, Morgan Kaufmann (1991) 23–35

[26] Radcliffe, N.J., Surry, P.D.: Fitness Variance of Formae and Performance Prediction. In Whitley, L.D., Vose, M.D., eds.: Foundations of Genetic Algorithms III, San Francisco, CA, Morgan Kaufmann (1994) 51–72

[27] Dzubera, J., Whitley, L.D.: Advanced Correlation Analysis of Operators for the Traveling Salesman Problem. In Schwefel, H.P., Männer, R., eds.: Parallel Problem Solving from Nature III. Volume 866 of Lecture Notes in Computer Science., Dortmund, Germany, Springer-Verlag, Berlin, Germany (1994) 68–77

[28] Manderick, B., de Weger, M., Spiessens, P.: The Genetic Algorithm and the Structure of the Fitness Landscape. In Belew, R.K., Booker, L.B., eds.: Proceedings of the 4^{th} International Conference on Genetic Algorithms, San Mateo, CA, Morgan Kaufmann (1991) 143–150

[29] Aickelin, U., Dowsland, K.: Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. Journal of Scheduling **3** (2000) 139–153

[30] Cotta, C., Troya, J.: A hybrid genetic algorithm for the 0-1 multiple knapsack problem. In Smith, G., Steele, N., Albrecht, R., eds.: Artificial Neural Nets and Genetic Algorithms 3, Wien New York, Springer-Verlag (1998) 251–255

[31] Varela, R., Puente, J., Vela, C.R.: Some issues in chromosome codification for scheduling with genetic algorithms. In Castillo, L., Borrajo, D., Salido, M.A., Oddi, A., eds.: Planning, Scheduling and Constraint Satisfaction: From Theory to Practice. Volume 117 of Frontiers in Artificial Intelligence and Applications. IOS Press (2005) 1–10

[32] Radcliffe, N.J.: The algebra of genetic algorithms. Annals of Mathematics and Artificial Intelligence **10** (1994) 339–384

[33] Oğuz, C., Ercan, M.F.: A genetic algorithm for hybrid flow-shop scheduling with multiprocessor tasks. Journal of Scheduling **8** (2005) 323–351

[34] Cotta, C., Aldana, J., Nebro, A., Troya, J.: Hybridizing genetic algorithms with branch and bound techniques for the resolution of the TSP. In Pearson, D., Steele, N., Albrecht, R., eds.: Artificial Neural Nets and Genetic Algorithms 2, Wien New York, Springer-Verlag (1995) 277–280

[35] Cotta, C., Troya, J.M.: Embedding branch and bound within evolutionary algorithms. Applied Intelligence **18** (2003) 137–153

[36] Gallardo, J.E., Cotta, C., Fernández, A.J.: A memetic algorithm with bucket elimination for the still life problem. In Gottlieb, J., Raidl, G., eds.: Evolutionary Computation in Combinatorial Optimization. Volume 3906 of Lecture Notes in Computer Science., Berlin Heidelberg, Springer-Verlag (2006) 73–85

[37] Ibaraki, T.: Combination with dynamic programming. In Bäck, T., Fogel, D., Michalewicz, Z., eds.: Handbook of Evolutionary Computation. Oxford University Press, New York NY (1997) D3.4:1–2

[38] Maheswaran, R., Ponnambalam, S.G., Aranvidan, C.: A meta-heuristic approach to single machine scheduling problems. International Journal of Advanced Manufacturing Technology **25** (2005) 772–776

[39] Wang, L., Zheng, D.Z.: A modified genetic algorithm for job-shop scheduling. International Journal of Advanced Manufacturing Technology **20** (2002) 72–76

[40] Liaw, C.F.: A hybrid genetic algorithm for the open shop scheduling problem. European Journal of Operational Research **124** (2000) 28–42

[41] França, P.M., Gupta, J., Mendes, A.S., Moscato, P., Veltnik, K.J.: Evolutionary algorithms for scheduling a flowshop manufacturing cell with sequence dependent family setups. Computers and Industrial Engineering **48** (2005) 491–506

[42] Hansen, P., Mladenović, N.: Variable neighborhood search: Principles and applications. European Journal of Operational Research **130** (2001) 449–467

[43] Cotta, C.: Scatter search with path relinking for phylogenetic inference. European Journal of Operational Research **169** (2006) 520–532

[44] Yeh, W.C.: A memetic algorithm fo the $n/2/\text{Flowshop}/\alpha\text{F}+\beta\text{C}_{max}$ scheduling problem. International Journal of Advanced Manufacturing Technology **20** (2002) 464–473

[45] Laguna, M., Martí, R.: Scatter Search. Methodology and Implementations in C. Kluwer Academic Publishers, Boston MA (2003)

[46] Nagata, Y., Kobayashi, S.: Edge assembly crossover: A high-power genetic algorithm for the traveling salesman problem. In Bäck, T., ed.: Proceedings of the Seventh International Conference on Genetic Algorithms, East Lansing, EE.UU., San Mateo, CA, Morgan Kaufmann (1997) 450–457

[47] Burke, E.K., Cowling, P.I., De Causmaecker, P., van den Berghe, G.: A memetic approach to the nurse rostering problem. Applied Intelligence **15** (2001) 199–214

[48] Yamada, T., Reeves, C.R.: Solving the C_{sum} permutation flowshop scheduling problem by genetic local search. In: 1998 IEEE International Conference on Evolutionary Computation, Piscataway, NJ, IEEE Press (1998) 230–234

[49] Cotta, C.: Memetic algorithms with partial lamarckism for the shortest common supersequence problem. In Mira, J., Álvarez, J., eds.: Artificial Intelligence and Knowledge Engineering Applications: a Bioinspired Approach. Volume 3562 of Lecture Notes in Computer Science., Berlin Heidelberg, Springer-Verlag (2005) 84–91

[50] Houck, C., Joines, J.A., Kay, M.G., Wilson, J.R.: Empirical investigation of the benefits of partial lamarckianism. Evolutionary Computation **5** (1997) 31–60

[51] Ishibuchi, H., Yoshida, T., Murata, T.: Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. IEEE Transactions on Evolutionary Computation **7** (2003) 204–223

[52] Jones, T.C., Forrest, S.: Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In Eshelman, L.J., ed.: Proceedings of the 6^{th} International Conference on Genetic Algorithms, Morgan Kaufmann (1995) 184–192

[53] Merz, P., Freisleben, B.: Fitness landscapes and memetic algorithm design. In Corne, D., Dorigo, M., Glover, F., eds.: New Ideas in Optimization. McGraw-Hill, Maidenhead, Berkshire, England, UK (1999) 245–260

[54] Bierwirth, C., Mattfeld, D.C., Watson, J.P.: Landscape regularity and random walks for the job shop scheduling problem. In Gottlieb, J., Raidl, G.R., eds.: Evolutionary Computation in Combinatorial Optimization. Volume 3004 of Lecture Notes in Computer Science., Berlin, Springer-Verlag (2004) 21–30

[55] Cotta, C., Fernández, A.: Analyzing fitness landscapes for the optimal golomb ruler problem. In Gottlieb, J., Raidl, G., eds.: Evolutionary Computation in Combinatorial Optimization. Volume 3248 of Lecture Notes in Computer Science., Berlin, Springer-Verlag (2005) 68–79

[56] Grefenstette, J.J.: Genetic algorithms for changing environments. In Männer, R., Manderick, B., eds.: Parallel Problem Solving from Nature II, Amsterdam, North-Holland Elsevier (1992) 137–144

[57] Hadj-Alouane, A.B., Bean, J.C., Murty, K.G.: A hybrid genetic/optimization algorithm for a task allocation problem. Journal of Scheduling **2** (1999) 181–201

[58] Tomassini, M.: Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time. Springer-Verlag (2005)

[59] Berretta, R., Cotta, C., Moscato, P.: Enhancing the performance of memetic algorithms by using a matching-based recombination algorithm: Results on the number partitioning problem. In Resende, M., Pinho de Sousa, J., eds.: Metaheuristics: Computer-Decision Making. Kluwer Academic Publishers, Boston MA (2003) 65–90

[60] Mendes, A., Cotta, C., Garcia, V., França, P., Moscato, P.: Gene ordering in microarray data using parallel memetic algorithms. In Skie, T., Yang, C.S., eds.: Proceedings of the 2005 International Conference on Parallel Processing Workshops, Oslo, Norway, IEEE Press (2005) 604–611

[61] Moscato, P., Mendes, A., Cotta, C.: Scheduling and production & control. In Onwubolu, G.C., Babu, B.V., eds.: New Optimization Techniques in Engineering. Springer-Verlag, Berlin Heidelberg (2004) 655–680

[62] Cotta, C., Alba, E., Troya, J.M.: Stochastic reverse hillclimbing and iterated local search. In: Proceedings of the 1999 Congress on Evolutionary Computation, Washington D.C., IEEE Neural Network Council - Evolutionary Programming Society - Institution of Electrical Engineers (1999) 1558–1565

[63] Cobb, H.G.: An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. Technical Report AIC-90-001, Naval Research Laboratory, Washington DC (1990)

[64] Merz, P., Freisleben, B.: Fitness Landscapes, Memetic Algorithms and Greedy Operators for Graph Bi-Partitioning. Evolutionary Computation **8** (2000) 61–91

[65] Yeh, W.C.: A memetic algorithm for the min k-cut problem. Control and Intelligent Systems **28** (2000) 47–55

[66] Berretta, R., Moscato, P.: The number partitioning problem: An open challenge for evolutionary computation ? In Corne, D., Dorigo, M., Glover,

F., eds.: New Ideas in Optimization. McGraw-Hill, Maidenhead, Berkshire, England, UK (1999) 261–278

[67] Aggarwal, C., Orlin, J., Tai, R.: Optimized crossover for the independent set problem. Operations Research **45** (1997) 226–234

[68] Hifi, M.: A genetic algorithm-based heuristic for solving the weighted maximum independent set and some equivalent problems. Journal of the Operational Research Society **48** (1997) 612–622

[69] Reeves, C.: Hybrid genetic algorithms for bin-packing and related problems. Annals of Operations Research **63** (1996) 371–396

[70] Coll, P., Durán, G., Moscato, P.: On worst-case and comparative analysis as design principles for efficient recombination operators: A graph coloring case study. In Corne, D., Dorigo, M., Glover, F., eds.: New Ideas in Optimization. McGraw-Hill, Maidenhead, Berkshire, England, UK (1999) 279–294

[71] Dorne, R., Hao, J.: A new genetic local search algorithm for graph coloring. In Eiben, A., Bäck, T., Schoenauer, M., Schwefel, H.P., eds.: Parallel Problem Solving From Nature V. Volume 1498 of Lecture Notes in Computer Science., Berlin, Springer-Verlag (1998) 745–754

[72] Beasley, J., Chu, P.: A genetic algorithm for the set covering problem. European Journal of Operational Research **94** (1996) 393–404

[73] França, P., Mendes, A., Moscato, P.: Memetic algorithms to minimize tardiness on a single machine with sequence-dependent setup times. In: Proceedings of the 5th International Conference of the Decision Sciences Institute, Athens, Greece, Atlanta, GA, USA, Decision Sciences Institute (1999) 1708–1710

[74] Miller, D., Chen, H., Matson, J., Liu, Q.: A hybrid genetic algorithm for the single machine scheduling problem. Journal of Heuristics **5** (1999) 437–454

[75] Mendes, A., Muller, F., França, P., Moscato, P.: Comparing meta-heuristic approaches for parallel machine scheduling problems with sequence-dependent setup times. In: Proceedings of the 15th International Conference on CAD/CAM Robotics & Factories of the Future, Aguas de Lindoia, Brasil. Volume 1., Campinas, SP, Brazil, Technological Center for Informatics Foundation (1999) 1–6

[76] Min, L., Cheng, W.: Identical parallel machine scheduling problem for minimizing the makespan using genetic algorithm combined with simulated annealing. Chinese Journal of Electronics **7** (1998) 317–321

[77] Chu, P., Beasley, J.: A genetic algorithm for the generalised assignment problem. Computers & Operations Research **24** (1997) 17–23

[78] Beasley, J., Chu, P.: A genetic algorithm for the multidimensional knapsack problem. Journal of Heuristics **4** (1998) 63–86

[79] Gottlieb, J.: Permutation-based evolutionary algorithms for multidimensional knapsack problems. In Carroll, J., Damiani, E., Haddad, H., Oppenheim, D., eds.: ACM Symposium on Applied Computing 2000, ACM Press (2000) 408–414

[80] Taguchi, T., Yokota, T., Gen, M.: Reliability optimal design problem with interval coefficients using hybrid genetic algorithms. Computers & Industrial Engineering **35** (1998) 373–376

[81] Merz, P., Freisleben, B.: A Comparison of Memetic Algorithms, Tabu Search, and Ant Colonies for the Quadratic Assignment Problem. In Angeline, P., ed.: 1999 Congress on Evolutionary Computation (CEC'99), Piscataway, NJ, USA, IEEE Press (1999) 2063–2070

[82] Levine, D.: A parallel genetic algorithm for the set partitioning problem. In Osman, I., Kelly, J., eds.: Meta-Heuristics: Theory & Applications. Kluwer Academic Publishers, Boston, MA, USA (1996) 23–35

[83] Holstein, D., Moscato, P.: Memetic algorithms using guided local search: A case study. In Corne, D., Dorigo, M., Glover, F., eds.: New Ideas in Optimization. McGraw-Hill, Maidenhead, Berkshire, England, UK (1999) 235–244

[84] Merz, P.: A comparison of memetic recombination operators for the traveling salesman problem. In Langdon, W., et al., eds.: GECCO, Morgan Kaufmann (2002) 472–479

[85] Ozcan, E., Mohan, C.: Steady state memetic algorithm for partial shape matching. In Porto, V., Saravanan, N., Waagen, D., eds.: Evolutionary Programming VII. Volume 1447 of Lecture Notes in Computer Science., Springer, Berlin (1998) 527–536

[86] Merz, P., Freisleben, B.: On the Effectiveness of Evolutionary Search in High–Dimensional NK-Landscapes. In Fogel, D., ed.: Proceedings of the 1998 IEEE International Conference on Evolutionary Computation, Piscataway, NJ, USA, IEEE Press (1998) 741–745

[87] Crain, T., Bishop, R., Fowler, W., Rock, K.: Optimal interplanetary trajectory design via hybrid genetic algorithm/recursive quadratic program search. In: Ninth AAS/AIAA Space Flight Mechanics Meeting, Breckenridge CO (1999) 99–133

[88] Kassotakis, I., Markaki, M., Vasilakos, A.: A hybrid genetic approach for channel reuse in multiple access telecommunication networks. IEEE Journal on Selected Areas in Communications **18** (2000) 234–243

[89] Raidl, G., Julstron, B.: A weighted coding in a genetic algorithm for the degree-constrained minimum spanning tree problem. In Carroll, J., Damiani, E., Haddad, H., Oppenheim, D., eds.: ACM Symposium on Applied Computing 2000, ACM Press (2000) 440–445

[90] Hopper, E., Turton, B.: A genetic algorithm for a 2d industrial packing problem. Computers & Industrial Engineering **37** (1999) 375–378

[91] Krzanowski, R., Raper, J.: Hybrid genetic algorithm for transmitter location in wireless networks. Computers, Environment and Urban Systems **23** (1999) 359–382

[92] Rodrigues, A., Ferreira, J.S.: Solving the rural postman problem by memetic algorithms. In de Sousa, J.P., ed.: Proceedings of the 4th Metaheuristic International Conference (MIC'2001), Porto, Portugal, July 16-20, 2001. (2001) 679–684

[93] Gen, M., Ida, K., Yinzhen, L.: Bicriteria transportation problem by hybrid genetic algorithm. Computers & Industrial Engineering **35** (1998) 363–366

[94] Novaes, A., De-Cursi, J., Graciolli, O.: A continuous approach to the design of physical distribution systems. Computers & Operations Research **27** (2000) 877–893

[95] Torres-Velazquez, R., Estivill-Castro, V.: A memetic algorithm instantiated with selection sort consistently finds global optima for the error-correcting graph isomorphism. In Yao, X., ed.: Proceedings of the IEEE 2002 Congress on Evolutionary Computation, CEC'02, May 12-17, 2002, Honolulu, Hawaii, USA. (2002) 1958–1963

[96] Kersting, S., Raidl, G., Ljubić, I.: A memetic algorithm for vertex-biconnectivity augmentation. In Cagnoni, S., et al., eds.: Applications of Evolutionary Computing, Proceedings of EvoWorkshops2002: EvoCOP, EvoIASP, EvoSTim. Volume 2279 of LNCS., Kinsale, Ireland, Springer-Verlag (2002) 101–110

[97] Merz, P., Zell, A.: Clustering gene expression profiles with memetic algorithms. In: 7th International Conference on Parallel Problem Solving from Nature - PPSN VII, September 7-11, 2002, Granada, Spain. (2002)

[98] Buriol, L., Resende, M., Ribeiro, C., Thorup, M.: A memetic algorithm for OSPF routing. In: Sixth INFORMS Telecommunications Conference,

March 10-13, 2002 Hilton Deerfield Beach, Boca Raton, Florida. (2002) 187–188

[99] Cotta, C., Fernández, A.: A hybrid GRASP - evolutionary algorithm approach to golomb ruler search. In Yao, X., et al., eds.: Parallel Problem Solving From Nature VIII. Volume 3242 of Lecture Notes in Computer Science., Berlin, Springer-Verlag (2004) 481–490

[100] Cotta, C., Dotú, I., Fernández, A.J., Hentenryck, P.V.: A memetic approach to golomb rulers. In Runarsson, T., et al., eds.: Parallel Problem Solving from Nature IX. Volume 4193 of Lecture Notes in Computer Science., Berlin Heidelberg, Springer-Verlag (2006) 252–261

[101] Gallardo, J.E., Cotta, C., Fernández, A.J.: A multi-level memetic/exact hybrid algorithm for the still life problem. In Runarsson, T., et al., eds.: Parallel Problem Solving from Nature IX. Volume 4193 of Lecture Notes in Computer Science., Berlin Heidelberg, Springer-Verlag (2006) 212–221

[102] dos Santos Coelho, L., Rudek, M., Junior, O.C.: Fuzzy-memetic approach for prediction of chaotic time series and nonlinear identification. In: 6th On-line World Conference on Soft Computing in Industrial Applications, Organized by World Federation of Soft Computing. (2001) Co-sponsored by IEEE Systems, Man, and Cybenertics Society.

[103] Burke, E., Smith, A.: A memetic algorithm to schedule grid maintenance. In: Proceedings of the International Conference on Computational Intelligence for Modelling Control and Automation, Vienna: Evolutionary Computation and Fuzzy Logic for Intelligent Control, Knowledge Acquisition and Information Retrieval, IOS Press (1999) 122–127

[104] Burke, E., Smith, A.: A multi-stage approach for the thermal generator maintenance scheduling problem. In: Proceedings of the 1999 Congress on Evolutionary Computation, Piscataway, NJ, USA, IEEE (1999) 1085–1092

[105] Cheng, R., Gen, M., Tsujimura, Y.: A tutorial survey of job-shop scheduling problems using genetic algorithms. II. Hybrid genetic search strategies. Computers & Industrial Engineering **37** (1999) 51–55

[106] Sevaux, M., Jouglet, A., Oğuz, C.: Combining constraint programming and memetic algorithm for the hybrid flowshop scheduling problem. In: ORBEL 19^{th} annual conference of the SOGESCI-BVWB, Louvain-la-Neuve, Belgium (2005)

[107] Ramat, E., Venturini, G., Lente, C., Slimane, M.: Solving the multiple resource constrained project scheduling problem with a hybrid genetic al-

gorithm. In Bäck, T., ed.: Proceedings of the Seventh International Conference on Genetic Algorithms, San Francisco CA, Morgan Kaufmann (1997) 489–496

[108] Ozdamar, L.: A genetic algorithm approach to a general category project scheduling problem. IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews) **29** (1999) 44–59

[109] Watson, J., Rana, S., Whitley, L., Howe, A.: The impact of approximate evaluation on the performance of search algorithms for warehouse scheduling. Journal of Scheduling **2** (1999) 79–98

[110] Dellaert, N., Jeunet, J.: Solving large unconstrained multilevel lot-sizing problems using a hybrid genetic algorithm. International Journal of Production Research **38** (2000) 1083–1099

[111] Ming, X., Mak, K.: A hybrid hopfield network-genetic algorithm approach to optimal process plan selection. International Journal of Production Research **38** (2000) 1823–1839

[112] Burke, E.K., Elliman, D.G., Weare, R.F.: A hybrid genetic algorithm for highly constrained timetabling problems. In: Proceedings of the Sixth International Conference on Genetic Algorithms, Morgan Kaufmann, San Francisco, CA (1995) 605–610

[113] Paechter, B., Cumming, A., Norman, M., Luchian, H.: Extensions to a Memetic timetabling system. In Burke, E., Ross, P., eds.: The Practice and Theory of Automated Timetabling. Volume 1153 of Lecture Notes in Computer Science. Springer Verlag (1996) 251–265

[114] de Causmaecker, P., van den Berghe, G., Burke, E.: Using tabu search as a local heuristic in a memetic algorithm for the nurse rostering problem. In: Proceedings of the Thirteenth Conference on Quantitative Methods for Decision Making, Brussels, Belgium (1999) abstract only, poster presentation

[115] Burke, E.K., De Causmaecker, P., van den Berghe, G.: Novel metaheuristic approaches to nurse rostering problems in belgian hospitals. In Leung, J., ed.: Handbook of Scheduling: Algorithms, Models, and Performance Analysis. Chapman Hall/CRC Press (2004) 44.1–44.18

[116] Costa, D.: An evolutionary tabu search algorithm and the NHL scheduling problem. INFOR **33** (1995) 161–178

[117] Burke, E.K., Newall, J.: A multi-stage evolutionary algorithm for the timetable problem. IEEE Transactions on Evolutionary Computation **3** (1999) 63–74

[118] Gonçalves, J.: A memetic algorithm for the examination timetabling problem. In: Optimization 2001, Aveiro, Portugal, July 23-25, 2001. (2001)

[119] Ichimura, T., Kuriyama, Y.: Learning of neural networks with parallel hybrid GA using a Royal Road function. In: 1998 IEEE International Joint Conference on Neural Networks. Volume 2., New York, NY, IEEE (1998) 1131–1136

[120] Topchy, A., Lebedko, O., Miagkikh, V.: Fast learning in multilayered networks by means of hybrid evolutionary and gradient algorithms. In: Proceedings of International Conference on Evolutionary Computation and its Applications. (1996) 390–398

[121] Aguilar, J., Colmenares, A.: Resolution of pattern recognition problems using a hybrid genetic/random neural network learning algorithm. Pattern Analysis and Applications **1** (1998) 52–61

[122] Krishna, K., Narasimha-Murty, M.: Genetic k-means algorithm. IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics) **29** (1999) 433–439

[123] Mignotte, M., Collet, C., Pérez, P., Bouthemy, P.: Hybrid genetic optimization and statistical model based approach for the classification of shadow shapes in sonar imagery. IEEE Transactions on Pattern Analysis and Machine Intelligence **22** (2000) 129–141

[124] Ostermark, R.: A neuro-genetic algorithm for heteroskedastic time-series processes: empirical tests on global asset returns. Soft Computing **3** (1999) 206–220

[125] Cotta, C., Troya, J.: Using a hybrid evolutionary-A* approach for learning reactive behaviors. In Cagnoni, S., et al., eds.: Real-World Applications of Evolutionary Computation. Volume 1803 of Lecture Notes in Computer Science., Edinburgh, Springer-Verlag (2000) 347–356

[126] Pratihar, D., Deb, K., Ghosh, A.: Fuzzy-genetic algorithms and mobile robot navigation among static obstacles. In: Proceedings of the 1999 Congress on Evolutionary Computation, Washington D.C., IEEE (1999) 327–334

[127] Ridao, M., Riquelme, J., Camacho, E., Toro, M.: An evolutionary and local search algorithm for planning two manipulators motion. In Del Pobil, A., Mira, J., Ali, M., eds.: Tasks and Methods in Applied Artificial Intelligence. Volume 1416 of Lecture Notes in Computer Science. Springer-Verlag, Berlin Heidelberg (1998) 105–114

[128] Chaiyaratana, N., Zalzala, A.: Hybridisation of neural networks and genetic algorithms for time-optimal control. In: Proceedings of the 1999 Congress on Evolutionary Computation, Washington D.C., IEEE (1999) 389–396

[129] Areibi, S., Moussa, M., Abdullah, H.: A comparison of genetic/memetic algorithms and heuristic searching. In: Proceedings of the 2001 International Conference on Artificial Intelligence ICAI 2001, Las Vegas, Nevada, June 25. (2001)

[130] Yeh, I.: Hybrid genetic algorithms for optimization of truss structures. Computer Aided Civil and Infrastructure Engineering **14** (1999) 199–206

[131] Pacey, M., Patterson, E., James, M.: A photoelastic technique for characterising fatigue crack closure and the effective stress intensity factor. Zeszyty Naukowe Politechniki Opolskiej, Seria: Mechanika **z.67** (2001) *VII Summer School of Fracture Mechanic, Current Research in Fatigue and Fracture*, Pokrzywna (Poland), 18-22 Jun. 2001.

[132] Wang, L., Yen, J.: Extracting fuzzy rules for system modeling using a hybrid of genetic algorithms and kalman filter. Fuzzy Sets and Systems **101** (1999) 353–362

[133] Zelinka, I., Vasek, V., Kolomaznik, K., Dostal, P., Lampinen, J.: Memetic algorithm and global optimization of chemical reactor. In: PC Control 2001, 13th International Conference on Process Control, High Tatras, Slovakia. (2001)

[134] Knödler, K., Poland, J., Zell, A., Mitterer, A.: Memetic algorithms for combinatorial optimization problems in the calibration of modern combustion engines. In Langdon, W.B., et al., eds.: GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, New York, Morgan Kaufmann Publishers (2002) 687

[135] Bos, A.: Aircraft conceptual design by genetic/gradient-guided optimization. Engineering Applications of Artificial Intelligence **11** (1998) 377–382

[136] Quagliarella, D., Vicini, A.: Hybrid genetic algorithms as tools for complex optimisation problems. In Blonda, P., Castellano, M., Petrosino, A., eds.: New Trends in Fuzzy Logic II. Proceedings of the Second Italian Workshop on Fuzzy Logic, Singapore, World Scientific (1998) 300–307

[137] Hodgson, R.: Memetic algorithm approach to thin-film optical coating design. In Hart, W., Krasnogor, N., Smith, J., eds.: Second Workshop on Memetic Algorithms (2nd WOMA), San Francisco, California, USA (2001) 152–157

[138] Srinivasan, D., Cheu, R., Poh, Y., Ng, A.: Development of an intelligent technique for traffic network incident detection. Engineering Applications of Artificial Intelligence **13** (2000) 311–322

[139] Urdaneta, A., Gómez, J., Sorrentino, E., Flores, L., Díaz, R.: A hybrid genetic algorithm for optimal reactive power planning based upon successive linear programming. IEEE Transactions on Power Systems **14** (1999) 1292–1298

[140] Haas, O., Burnham, K., Mills, J.: Optimization of beam orientation in radiotherapy using planar geometry. Physics in Medicine and Biology **43** (1998) 2179–2193

[141] Wehrens, R., Lucasius, C., Buydens, L., Kateman, G.: HIPS, A hybrid self-adapting expert system for nuclear magnetic resonance spectrum interpretation using genetic algorithms. Analytica Chimica ACTA **277** (1993) 313–324

[142] Li, F., Morgan, R., Williams, D.: Economic environmental dispatch made easy with hybrid genetic algorithms. In: Proceedings of the International Conference on Electrical Engineering. Volume 2., Beijing, China, Int. Acad. Publishers (1996) 965–969

[143] Ostermark, R.: Solving irregular econometric and mathematical optimization problems with a genetic hybrid algorithm. Computational Economics **13** (1999) 103–115

[144] Musil, M., Wilmut, M., Chapman, N.: A hybrid simplex genetic algorithm for estimating geoacoustic parameters using matched-field inversion. IEEE Journal of Oceanic Engineering **24** (1999) 358–369

[145] Reich, C.: Simulation of imprecise ordinary differential equations using evolutionary algorithms. In Carroll, J., Damiani, E., Haddad, H., Oppenheim, D., eds.: ACM Symposium on Applied Computing 2000, ACM Press (2000) 428–432

[146] Wei, P., Cheng, L.: A hybrid genetic algorithm for function optimization. Journal of Software **10** (1999) 819–823

[147] Wei, X., Kangling, F.: A hybrid genetic algorithm for global solution of nondifferentiable nonlinear function. Control Theory & Applications **17** (2000) 180–183

[148] Cadieux, S., Tanizaki, N., Okamura, T.: Time efficient and robust 3-D brain image centering and realignment using hybrid genetic algorithm. In: Proceedings of the 36th SICE Annual Conference, IEEE (1997) 1279–1284

[149] Krishna, K., Ramakrishnan, K., Thathachar, M.: Vector quantization using genetic k-means algorithm for image compression. In: 1997 International Conference on Information, Communications and Signal Processing. Volume 3., New York, NY, IEEE (1997) 1585–1587

[150] Yoneyama, M., Komori, H., Nakamura, S.: Estimation of impulse response of vocal tract using hybrid genetic algorithm-a case of only glottal source. Journal of the Acoustical Society of Japan **55** (1999) 821–830

[151] Merz, P., Zell, A.: Clustering gene expression profiles with memetic algorithms. In Yao, X., et al., eds.: Parallel Problem Solving From Nature VIII. Volume 3242 of Lecture Notes in Computer Science., Berlin, Springer-Verlag (2004) 811–820

[152] Moscato, P., Berretta, R., Mendes, A.: A new memetic algorithm for ordering datasets: Applications in microarray analysis. In Doerner, K., et al., eds.: Proceedings of the 6th Metaheuristics International Conference, Vienna, Austria (2005) 695–700

[153] Knowles, J., Corne, D.: M-PAES: A memetic algorithm for multiobjective optimization. In: Proceedings of the 2000 Congress on Evolutionary Computation CEC00, La Jolla Marriott Hotel La Jolla, California, USA, IEEE Press (2000) 325–332

[154] Ponnambalam, S.G., Mohan Reddy, M.: A GA-SA multiobjective hybrid search algorithm for integrating lot sizing and sequencing in flow-line scheduling. International Journal of Advanced Manufacturing Technology **21** (2003) 126–137

[155] Krasnogor, N., Smith, J.E.: Emergence of profitable search strategies based on a simple inheritance mechanism. In Spector, L., et al., eds.: Proceedings of the 2001 Genetic and Evolutionary Computation Conference, Morgan Kaufmann (2001) 432–439

[156] Li, J., Kwan, R.S.K.: A self adjusting algorithm for driver scheduling. Journal of Heuristics **11** (2005) 351–367

[157] Cowling, P.I., Ouelhadj, D., Petrovic, S.: Dynamic scheduling of steel casting and milling using multi-agents. Production Planning and Control **15** (2002) 1–11

[158] Cowling, P.I., Ouelhadj, D., Petrovic, S.: A multi-agent architecture for dynamic scheduling of steel hot rolling. Journal of Intelligent Manufacturing **14** (2002) 457–470

[159] Burke, E.K., Kendall, G., Newall, J., Hart, E., Ross, P., Schulenburg, S.: Hyperheuristics: an emerging direction in modern search technology. In Glover, F., Kochenberger, G., eds.: Handbook of Metaheuristics. Kluwer Academic Publishers, Boston MA (2003) 457–474

[160] Kendall, G., Soubeiga, E., Cowling, P.I.: Choice function and random hyperheuristics. In: Proceedings of the 4^{th} Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'02). (2002) 667–671

[161] Downey, R., Fellows, M.: Parameterized Complexity. Springer-Verlag (1998)

Búsquedas Multiarranque*

Abraham Duarte[a], **Rafael Martí**[b], **J. Marcos Moreno-Vega**[c]

[a] Dpto. de Informática, Estadística y Telemática
Escuela Superior de Ciencias Experimentales y Tecnología
Universidad Rey Juan Carlos
abraham.duarte@urjc.es

[b] Dpto. de Estadística e Investigación Operativa
Facultad de Matemáticas
Universidad de Valencia
rafael.marti@uv.es

[c] Dpto. de Estadística, I.O. y Computación
Escuela Técnica Superior de Ingeniería Informática
Universidad de La Laguna
jmmoreno@ull.es

1 Introducción

Un problema de *Optimización Combinatoria* puede formularse como:

$$optimizar_{X \in S} f(X)$$

donde $f : S \rightarrow \mathbb{R}$ es una función que a cada $X \in S$ asocia un número real, y S es un conjunto finito o infinito numerable de puntos. Al conjunto S se le conoce como *espacio de soluciones* o *región factible* y a la función f por *función de costo* o *función objetivo*. Por optimizar se entiende minimizar o maximizar la función objetivo f sobre el espacio solución. Un problema queda caracterizado por el espacio de soluciones y por la función objetivo, por lo que, a partir de ahora, lo denotaremos por el par (S, f).

*Este trabajo ha sido parcialmente financiado por el Ministerio de Ciencia y Tecnología, a través de los proyectos TIN 2005-08943-C02-02, TIN2006-02696 y TIN2005-08404-C04-03 (70% son fondos FEDER), y por el Gobierno de Canarias, a través del proyecto PI042004/088. La actividad desarrollada se enmarca dentro de los objetivos de la red RedHeur (proyecto TIN2004-20061-E).

Existen multitud de problemas reales que pueden formularse como un problema de optimización combinatoria. A esta clase pertenecen, por ejemplo, algunos problemas de localización de servicios, de determinación de rutas óptimas o de asignación de recursos.

Las Búsquedas Locales son los procedimientos más simples para resolver este tipo de problemas. En una Búsqueda Local, dada una solución inicial del problema, se realizan movimientos de mejora mientras sea posible. Un movimiento es una modificación de una solución que suministra otra solución del problema. Las Búsquedas Locales no suministran, en general, la solución óptima del problema. Para aumentar la probabilidad de encontrar dicha solución, puede aplicarse una Búsqueda Local a varias soluciones de la región factible. Este método recibe el nombre de Búsqueda Multiarranque.

En el presente trabajo se presentan los fundamentos de las Búsquedas Multiarranque para problemas combinatorios. Para ello, se describen algunas de las variantes propuestas para los diferentes elementos que definen estas búsquedas: mecanismo de generación de soluciones iniciales, método de mejora (búsqueda local) y regla de parada. El trabajo se estructura como sigue. En la próxima sección se introducen las búsquedas locales, en la sección 3 se describe la Búsqueda Multiarranque y se enumeran algunas de sus variantes y en la sección 4 se muestran dos Búsquedas Multiarranque para el Problema de la Máxima Diversidad. Por último, se listan las referencias bibliográficas que aparecen en el texto.

2 Búsqueda Local

En optimización real continua, un concepto importante es el de *óptimo local* de la función que se define como cualquier punto x^* para el que exista un entorno, en la topología usual de $\mathbb{R}$, de tal forma que x^* sea óptimo en ese entorno. La importancia radica en que el óptimo global puede definirse como aquel óptimo local con mejor valor de la función objetivo. Es decir, se dispone de una condición necesaria para encontrar el óptimo global de la función. Así, al menos en teoría, la dimensión del problema se reduce al tener que buscar la solución óptima del problema sólo entre los óptimos locales.

Para intentar usar estas ideas en los problemas discretos que nos ocupan, se define el entorno de una solución.

Definición 1
Dado el problema (S, f), una estructura de entorno es una función $N : S \to 2^S$ que asocia a cada solución $X \in S$ un conjunto $N(X) \subset S$ de soluciones cercanas a X en algún sentido. El conjunto $N(X)$ se llama entorno de la solución X, y cada $Y \in N(X)$, solución vecina de X.

La anterior definición es bastante general y deja a criterio del decisor el establecer cuando dos soluciones están *cercanas en algún sentido*.

Una estructura de entorno particularmente importante para problemas combinatorios es aquella en la que se sustituyen k elementos presentes en la solución por k elementos no presentes en la misma. Esta sustitución puede implementarse de varias formas que, básicamente, consisten en cambiar de valor k elementos de la solución. Así puede consistir en cambiar de 1 a 0 k posiciones de la solución y de 0 a 1 otras k posiciones distintas, sustituir k aristas presentes en la solución por otras k aristas, etc. A continuación se describe esta estructura de entorno para el problema de la p-mediana.

Problema de la p-mediana Dado un conjunto de puntos de demanda, $\mathcal{D} = \{d_1, \ldots, d_n\}$, y un conjunto de posibles localizaciones, $\mathcal{L} = \{l_1, \ldots, l_m\}$, se pretende determinar la ubicación óptima de p servicios que minimice la suma de los costos entre los puntos de demanda y los servicios.

Para formalizar el problema, se considera la existencia de una matriz $n \times m$ con el costo $c(d_i, l_j)$ que supone atender al punto de demanda d_i desde la localización l_j. La región factible de este problema está formada por subconjuntos de $\mathcal{L}$ con p puntos

$$S = \{X \subset \mathcal{L} : \mid X \mid = p\}.$$

Sea el costo que supone atender el punto de demanda d_i desde la solución X igual a

$$c(d_i, X) = \min_{l_j \in X} c(d_i, l_j).$$

El problema se formula como

$$\min_{X \in S} \sum_{i=1}^{n} c(d_i, X)$$

Las soluciones del problema de la p-mediana son subconjuntos de $\mathcal{L}$ de tamaño p, formados por los índices de las localizaciones en las que se ubican los servicios.

Ejemplo 1
En la estructura de entorno del k-intercambio, una solución, X', del problema de la p-mediana es vecina de otra solución X, si puede obtenerse de ésta intercambiando una localización presente en la solución por otra no presente en la misma. Así, si $m = 5$, $p = 3$ y $k = 1$ las soluciones $X = \{l_1, l_3, l_5\}$ y $X' = \{l_1, l_3, l_4\}$ son vecinas, ya que X' puede obtenerse desde X intercambiando l_5 por l_4. En la tabla 1 se muestran todas las soluciones factibles de un problema de la 3-mediana cuando $m = 5$, el valor objetivo de éstas (escogido de forma ficticia para este ejemplo) y las correspondientes soluciones vecinas si $k = 1$.

Una *Búsqueda Local* comienza con una *Solución Actual* de la región factible con costo asociado *Objetivo(Solución Actual)*. A continuación selecciona, si es posible,

Solución	Costo	Soluciones vecinas					
$\{l_1, l_2, l_3\}$	10	$\{l_1, l_2, l_4\}$	$\{l_1, l_2, l_5\}$	$\{l_1, l_3, l_4\}$	$\{l_1, l_3, l_5\}$	$\{l_2, l_3, l_4\}$	$\{l_2, l_3, l_5\}$
$\{l_1, l_2, l_4\}$	6	$\{l_1, l_2, l_3\}$	$\{l_1, l_2, l_5\}$	$\{l_1, l_3, l_4\}$	$\{l_1, l_4, l_5\}$	$\{l_2, l_3, l_4\}$	$\{l_2, l_4 l_5\}$
$\{l_1, l_2, l_5\}$	8	$\{l_1, l_2, l_3\}$	$\{l_1, l_2, l_4\}$	$\{l_1, l_3, l_5\}$	$\{l_1, l_4, l_5\}$	$\{l_3, l_2, l_5\}$	$\{l_2, l_4, l_5\}$
$\{l_1, l_3, l_4\}$	5	$\{l_1, l_2, l_4\}$	$\{l_1, l_4, l_5\}$	$\{l_1, l_2, l_3\}$	$\{l_1, l_3, l_5\}$	$\{l_1, l_2, l_4\}$	$\{l_1, l_4, l_5\}$
$\{l_1, l_3, l_5\}$	7	$\{l_1, l_2, l_5\}$	$\{l_1, l_4, l_5\}$	$\{l_1, l_2, l_3\}$	$\{l_1, l_3, l_4\}$	$\{l_2, l_3, l_5\}$	$\{l_3, l_4, l_5\}$
$\{l_1, l_4, l_5\}$	6	$\{l_1, l_2, l_4\}$	$\{l_1, l_3, l_4\}$	$\{l_1, l_2, l_5\}$	$\{l_1, l_3, l_5\}$	$\{l_2, l_4, l_5\}$	$\{l_3, l_4, l_5\}$
$\{l_2, l_3, l_4\}$	6	$\{l_1, l_2, l_4\}$	$\{l_2, l_4, l_5\}$	$\{l_1, l_3, l_4\}$	$\{l_3, l_4, l_5\}$	$\{l_1, l_2, l_3\}$	$\{l_2, l_3, l_5\}$
$\{l_2, l_3, l_5\}$	8	$\{l_1, l_2, l_5\}$	$\{l_2, l_4, l_5\}$	$\{l_1, l_3, l_5\}$	$\{l_3, l_4, l_5\}$	$\{l_1, l_2, l_3\}$	$\{l_2, l_3, l_4\}$
$\{l_2, l_4, l_5\}$	3	$\{l_1, l_4, l_5\}$	$\{l_3, l_4, l_5\}$	$\{l_1, l_2, l_5\}$	$\{l_2, l_3, l_5\}$	$\{l_1, l_2, l_4\}$	$\{l_2, l_3, l_4\}$
$\{l_3, l_4, l_5\}$	4	$\{l_1, l_4, l_5\}$	$\{l_2, l_4, l_5\}$	$\{l_1, l_3, l_5\}$	$\{l_2, l_3, l_5\}$	$\{l_1, l_3, l_4\}$	$\{l_2, l_3, l_4\}$

Tabla 1: Soluciones factibles y vecinas de un problema de la 3-mediana con $m = 5$ y $k = 1$

```
procedure Búsqueda Local(Var Solución Actual);
begin
   repeat
      Obtener(Solución Vecina/
              Objetivo(Solución Vecina) < Objetivo(Solución Actual));
      Solución Actual := Solución Vecina;
   until (Objetivo(Solución Vecina) ≥ Objetivo(Solución Actual),
         ∀ Solución Vecina);
end
```

Figura 1: Descripción general de la *Búsqueda Local*

una *Solución Vecina* con coste *Objetivo(Solución Vecina) < Objetivo(Solución Actual)*. Si tal solución existe, el algoritmo continúa con *Solución Vecina*. En caso contrario, el procedimiento termina con *Solución Actual* como solución propuesta por el algoritmo. El procedimiento puede describirse como aparece en la figura 1.

El principal inconveniente del procedimiento Búsqueda Local radica en que, en general, suministra soluciones localmente óptimas, con respecto a la estructura de entorno considerada, que pueden estar alejadas de la solución óptima global. Este hecho se pone de manifiesto en el siguiente ejemplo.

Ejemplo 2
Considérese el problema de localización la 2*-mediana Supóngase que el conjunto de posibles localizaciones consta de los cuatro puntos del cuadrado unidad:*

$$\mathcal{L} = \left\{ l_1 = \left(\frac{1}{4}, \frac{1}{2}\right), l_2 = \left(\frac{1}{2}, \frac{3}{4}\right), l_3 = \left(\frac{1}{2}, \frac{1}{4}\right), l_4 = \left(\frac{3}{4}, \frac{1}{2}\right) \right\}$$

la demanda está formada por los puntos

$$\mathcal{D} = \left\{ d_1 = \left(\frac{3}{8}, \frac{5}{8}\right), d_2 = \left(\frac{3}{8}, \frac{3}{8}\right), d_3 = \left(\frac{5}{8}, \frac{5}{8}\right), d_4 = \left(\frac{5}{8}, \frac{1}{2}\right), d_5 = \left(\frac{5}{8}, \frac{3}{8}\right) \right\}$$

y se pretende encontrar la 2*-mediana con distancia euclídea. La matriz de distancias es la siguiente:*

$$\begin{pmatrix} & d_1 & d_2 & d_3 & d_4 & d_5 \\ l_1 & \sqrt{\frac{1}{32}} & \sqrt{\frac{1}{32}} & \sqrt{\frac{10}{64}} & \frac{3}{8} & \sqrt{\frac{10}{64}} \\ l_2 & \sqrt{\frac{1}{32}} & \sqrt{\frac{10}{64}} & \sqrt{\frac{1}{32}} & \sqrt{\frac{5}{64}} & \sqrt{\frac{10}{64}} \\ l_3 & \sqrt{\frac{10}{64}} & \sqrt{\frac{1}{32}} & \sqrt{\frac{10}{64}} & \sqrt{\frac{5}{64}} & \sqrt{\frac{1}{32}} \\ l_4 & \sqrt{\frac{10}{64}} & \sqrt{\frac{10}{64}} & \sqrt{\frac{1}{32}} & \frac{1}{8} & \sqrt{\frac{10}{64}} \end{pmatrix}$$

Si se utiliza la estructura del 1*-intercambio, las posibles soluciones del problema, sus costos respectivos y las soluciones vecinas, así como los costos de éstas (entre paréntesis), se recogen en la tabla siguiente.*

Solución	*Costo*	*Vecinas*			
$X_1 = \{l_1, l_2\}$	1.205	$X_2(1.205)$	$X_3(0.832)$	$X_4(0.986)$	$X_5(1.05)$
$X_2 = \{l_1, l_3\}$	1.205	$X_1(1.205)$	$X_3(0.832)$	$X_4(0.986)$	$X_6(1.05)$
$X_3 = \{l_1, l_4\}$	0.832	$X_1(1.205)$	$X_2(1.205)$	$X_5(1.05)$	$X_6(1.05)$
$X_4 = \{l_2, l_3\}$	0.986	$X_1(1.205)$	$X_2(1.205)$	$X_5(1.05)$	$X_6(1.05)$
$X_5 = \{l_2, l_4\}$	1.050	$X_1(1.205)$	$X_3(0.832)$	$X_4(0.986)$	$X_6(1.05)$
$X_6 = \{l_3, l_4\}$	1.050	$X_2(1.205)$	$X_3(0.832)$	$X_4(0.986)$	$X_5(1.05)$

Si la solución actual del procedimiento búsqueda local es X_4*, el algoritmo acaba con esta solución, ya que todas las vecinas de* X_4 *poseen un costo mayor. Sin embargo,* X_4 *no es la solución óptima. El algoritmo se ha estancado en un mínimo local no global. Nótese además que la solución propuesta por el algoritmo puede estar muy alejada de la solución óptima del problema. En el ejemplo que tratamos, la solución propuesta no posee ninguna localización presente en la solución óptima.*

2.1 Muestreos en el entorno.

En la Búsqueda Local, dada la solución actual, se debe generar una nueva solución, del entorno de ésta, con coste menor. En muchas ocasiones, se intenta

una exploración completa del entorno de la solución actual en busca de la mejor vecina. Sin embargo, esta estrategia suele ser muy ineficiente. Por ello, deben considerarse otros muestreos del entorno.

1. *Búsqueda del mejor:* se realiza una búsqueda exhaustiva por el entorno de la solución y se toma la mejor vecina.

   Si se aplica una búsqueda local del mejor desde la solución $\{l_1, l_2, l_3\}$ con costo 10, el recorrido de la misma (ver cuadro 1) es $\{l_1, l_2, l_3\}$, $\{l_1, l_3, l_4\}$, $\{l_1, l_2, l_4\}$, $\{l_2, l_4, l_5\}$. Se obtiene la mejor solución del problema, $\{l_2, l_4, l_5\}$, con costo 3.

2. *Búsqueda del primer mejor:* se recorre en orden el entorno de la solución actual hasta encontrar una solución mejor.

   Si se aplica una búsqueda local del primer mejor desde la solución $\{l_1, l_2, l_3\}$ con costo 10, el recorrido de la misma (ver cuadro 1) es $\{l_1, l_2, l_3\}$, $\{l_1, l_2, l_4\}$, $\{l_1, l_3, l_4\}$. Se obtiene la solución $\{l_1, l_3, l_4\}$ con costo 5.

3. *Muestreo aleatorio:* se escoge aleatoriamente una solución del entorno de la solución actual. O la mejor de una muestra seleccionada aleatoriamente del entorno.

   Si se aplica una búsqueda local con muestreo aleatorio desde la solución $\{l_1, l_2, l_3\}$ con costo 10, el recorrido de la misma puede ser $\{l_1, l_2, l_3\}$, $\{l_2, l_3, l_4\}$, $\{l_3, l_4, l_5\}$, $\{l_2, l_4, l_5\}$). Se obtiene la mejor solución del problema, $\{l_2, l_4, l_5\}$, con costo 3. Nótese que, en este caso, puede obtenerse un recorrido distinto al comenzar una nueva búsqueda local desde la misma solución $\{l_1, l_2, l_3\}$.

4. *Muestreo heurístico:* se toma aquella solución (o equivalentemente se realiza aquel movimiento) que, con base a una evaluación heurística, suministre una solución mejor que la actual.

5. *Muestreo* Aspiration Plus: se muestrea el entorno hasta que se alcanza un valor mínimo de la función objetivo establecido previamente. Una vez alcanzado este nivel de aspiración, se analiza un número adicional de soluciones en busca de soluciones con mejor calidad. Para controlar el número de soluciones evaluadas en el entorno de cualquier solución, se establece el número mínimo y máximo de movimientos que pueden analizarse en cada iteración. Este muestreo ha sido propuesto por Glover [17].

3 Multiarranque

El principal inconveniente de las Búsquedas Locales es que, en general, suministran soluciones localmente óptimas que pueden estar muy alejadas (en términos

```
procedure Búsqueda con Arranque Múltiple
begin
    Generar (Solución Inicial);
    Mejor Solución := Solución Inicial;
    repeat
        Solución Actual := Búsqueda Local(Solución Inicial);
        if Objetivo(Solución Actual) < Objetivo(Mejor Solución)
            then Mejor Solución := Solución Actual;
        Generar(Solución Inicial);
    until criterio de parada;
end.
```

Figura 2: Descripción de la *Búsqueda con Arranque Múltiple.*

de valor objetivo) de la solución o soluciones óptimas globales. Una alternativa para solventar este inconveniente consiste en aplicar Búsquedas Locales desde varias soluciones de partida. La repetición de los procesos *Generar Solución Inicial* y *Búsqueda Local* constituye el primer Método Multiarranque descrito en la literatura [6] [28]. Este esquema puede generalizarse para contemplar diferentes Métodos Multiarranque que consisten en aplicar reiteradamente un optimizador o método de búsqueda desde diferentes soluciones iniciales.

La Figura 2 muestra el esquema general de un Método Multiarranque. Típicamente podemos hablar de dos fases en cada paso. En la primera, se construye una solución *Solución Inicial* y en la segunda se trata de mejorar mediante la aplicación de un método de búsqueda, obteniendo la solución *Solución Actual* (que eventualmente puede ser igual a *Solución Inicial*).

En algunas aplicaciones, la fase 1 se limita a la simple generación aleatoria de las soluciones, mientras que en otros ejemplos se emplean sofisticados métodos de construcción que consideran las características del problema de optimización para obtener soluciones iniciales de calidad. Algo similar ocurre con el método de búsqueda de la fase 2. Podemos encontrar algoritmos de Búsqueda Local que, a partir de la solución inicial, conducen al óptimo local más cercano mediante una serie de movimientos de mejora, o elaborados procedimientos metaheurísticos que realizan una búsqueda *inteligente* del espacio de soluciones y tratan de alcanzar la solución óptima del problema, evitando quedar atrapados en un óptimo local de baja calidad. En cuanto a la condición de parada, se han propuesto desde criterios simples, como el de parar después de un número dado de iteraciones, hasta criterios que analizan la evolución de la búsqueda y aseguran, en muchos casos, la convergencia asintótica al óptimo global del problema. Otra cuestión a considerar es si el método de búsqueda de la segunda fase debe aplicarse a todos los puntos generados o sólo a un subconjunto de dichos puntos. La combinación de todos estos elementos (construcción de soluciones iniciales, optimizador o método

de mejora, criterio de parada, ...) suministra una gran variedad de procedimientos híbridos basados en el esquema Multiarranque.

Martí [29] hace una revisión de los métodos multiarranque y propone una clasificación basada en el uso o no de memoria y en el grado de reconstrucción de la solución de inicio. Schoen [38] hace una revisión personal de tales métodos, dando una definición formal de los mismos y subrayando las características propias de estos métodos. En [31] se enumeran algunas de las alternativas propuestas para los diferentes elementos que definen un Método Multiarranque. En las siguientes subsecciones, describimos brevemente algunas de las variantes propuestas que pueden encontrarse en [31] .

3.1 Solución inicial.

Los mecanismos de generación de soluciones iniciales pueden ser dependientes o independientes del problema considerado. Entre los mecanismos dependientes del problema destacamos los de generación aleatoria, determinística y mixta. A continuación se describen estos tres métodos para el problema de la p-mediana.

- **Generación aleatoria.** Escoger aleatoriamente p localizaciones y establecer en ellas los servicios.

  Para aplicar el mecanismo de generación aleatoria al ejemplo 2, deben escogerse aleatoriamente dos servicios del conjunto de posibles localizaciones $\mathcal{L} = \{l_1, l_2, l_3, l_4\}$. Así, una de las soluciones que podrían obtenerse es $X_5 = \{l_2, l_4\}$.

- **Generación determinista.** En primer lugar, establecer un servicio en la localización más cercana al punto medio de los puntos de demanda. A continuación, establecer un servicio en la localización más alejada de los servicios previamente establecidos. Repetir el paso anterior hasta localizar p servicios.

  En el ejemplo 2, el punto medio de los puntos de demanda es $(0.525, 0.425)$. La localización más cercana a este punto es l_3, y la localización más alejada de l_3 es l_2. Por tanto, la solución inicial empleando la generación determinística es $X_4 = \{l_2, l_3\}$.

- **Generación mixta.** En primer lugar, establecer un servicio en la localización más cercana al punto medio de los puntos de demanda. A continuación, escoger aletoriamente el siguiente servicio con probabilidades proporcionales a la distancia a los servicios previamente establecidos. Repetir el paso anterior hasta localizar p servicios.

  En el ejemplo 2, el punto medio de los puntos de demanda es $(0.525, 0.425)$. La localización más cercana a este punto es l_3. Teniendo en cuenta la distancia que separa las localizaciones l_1, l_2 y l_4 de l_3, se sigue que existe

la misma probabilidad de escoger l_1 o l_4 como nuevo servicio, y que esta probabilidad es menor que la de escoger l_2. El nuevo servicio debe escogerse usando a estas probabilidades. Si suponemos que se escoge l_1, se obtiene la solución inicial $X_2 = \{l_1, l_3\}$.

Boese et al. [8] analizan la relación entre los óptimos locales de un problema al tratar de determinar el mejor de todos ellos. Basado en los resultados de ese estudio, proponen un método multiarranque, llamado *Adaptive Multistart* (AMS), en el que los puntos iniciales se generan a partir de los mejores óptimos locales encontrados. En un primer paso, AMS genera r soluciones al azar y les aplica a todas ellas un procedimiento de búsqueda local *greedy* para determinar el conjunto inicial de óptimos locales. En el segundo paso (*adaptive*) se construyen las soluciones iniciales a partir del conjunto de óptimos locales. A estas soluciones iniciales se les aplica varias veces el método de mejora. Los autores prueban el método en la resolución del problema del viajante de comercio, y muestran que mejora significativamente a implementaciones previas de métodos multiarranque.

Hagen y Kang [19] proponen un método multiarranque de tipo AMS para el problema de partición VLSI donde el objetivo es minimizar el número de señales que circulan entre componentes. El método tiene dos fases. En la primera se genera un conjunto de soluciones aleatorias y se les aplica a todas ellas un algoritmo de búsqueda local, obteniendo un conjunto de óptimos locales. En la segunda parte, se construyen soluciones iniciales como los puntos centrales de los mejores óptimos locales conocidos. Con el objetivo de reducir el tamaño del problema a resolver, se añade una fase de preproceso basada en técnicas de agrupamiento. Un estudio empírico permite establecer la superioridad del método propuesto frente a algoritmos previos para este problema.

Uno de los métodos multiarranque más aplicados actualmente es el denominado GRASP, debido a Feo y Resende [7] [8]. Como el resto de métodos multiarranque, GRASP consta de dos fases. La construcción o generación y la mejora o búsqueda. En cada iteración de la fase constructiva, GRASP mantiene un conjunto de elementos candidatos que pueden ser añadidos a la solución parcial que se está construyendo. Todos los elementos candidatos se evalúan usando una función que mide su atractivo. En lugar de seleccionar el mejor de todos los elementos, se construye la lista restringida de candidatos RCL (*restricted candidate list*) con los mejores según una cantidad establecida (ésta es la parte *greedy* del método). El elemento que finalmente se añade a la solución parcial actual, se escoge al azar del conjunto RCL (ésta es la parte probabilística). Entonces se recalcula la lista de elementos candidatos y se realiza una nueva iteración (ésta es la parte *que se adapta* en el método). Estos pasos se reiteran hasta que se obtiene una solución del problema. A ésta se le aplica el método de mejora (ésta es la parte de búsqueda del método). A continuación, se repiten la fase constructiva y de mejora hasta que se cumpla el criterio de parada.

Existen diferentes variantes de este esquema entre las que podemos destacar el método heurístico conocido como semi-greedy debido a Hart y Shogan [21]. Este método sigue el esquema multiarranque basado en aleatorizar una evaluación greedy en la construcción, pero no tiene una fase de búsqueda local o mejora. Actualmente se están implementando versiones de este método denominadas *Reactive GRASP* en donde el ajuste de los parámetros necesarios (básicamente los que determinan la RCL) se realiza de forma dinámica según el estado de la búsqueda [36].

Fleurent y Golver [14] proponen un método multiarranque AMP (*Adaptive Memory Programming*) en el que se modifica la función de evaluación usando una medida de frecuencia para intensificar la búsqueda en un procedimiento constructivo. Los autores aplican su método al problema de asignación cuadrática y muestran la ventaja de usar tal estructura de memoria.

Melián et al. [33] usan información sobre las soluciones de inicio y sobre los óptimos locales encontrados para dirigir la búsqueda. La técnica se diseña para problemas combinatorios en los que se desea encontrar la mejor selección de un número dado de ítems desde un universo. Sea $RefSet$ el conjunto de óptimos locales encontrados hasta el momento. Para cada $x_i^{'}$ del conjunto $RefSet$, se conoce el porcentaje de búsquedas locales que, comenzando en soluciones a distancia menor o igual que k ($k = 1, \ldots$) de $x_i^{'}$, acaban en $x_i^{'}$. Sea $k(x_i^{'})$ el valor de k cuyo porcentaje asociado es mayor que α (parámetro fijado por el usuario), y considérese

$$k^* = \max_{x_i^{'} \in RefSet} k(x_i^{'}).$$

Sea, además, $w(x_i^{'})$ el porcentaje de búsquedas locales que acaban en $x_i^{'}$, y, para cada item u,

$$w(u) = \sum_{u \in x_i^{'} \in RefSet} w(x_i^{'}),$$

la suma de los porcentajes de aquellos óptimos locales que incluyen a u como parte de la solución.

Los anteriores valores se emplean para diversificar e intensificar la búsqueda. Para diversificar la misma, se desarrollan búsquedas locales desde soluciones a distancia mayor que k^* de un óptimo local seleccionado aleatoriamente de $RefSet$. Para intensificar la búsqueda, se construye aleatoriamente una solución en la que la probabilidad de incluir un elemento u en la misma es proporcional a los valores $w(u)$.

3.2 Reglas de parada

Como ocurre con cualquier método heurístico para resolver un problema, uno de los elementos más difíciles de fijar en un Método Multiarranque es el criterio

de parada. Los principales criterios de parada propuestos analizan tres variables aleatorias: valores objetivos de los mínimos locales, número de mínimos locales distintos de la función objetivo y número de iteraciones necesarias para alcanzar el mínimo global.

Los y Lardinois [28], en uno de los primeros trabajos dedicados a los Métodos Multiarranque, obtienen reglas de parada a partir de la función de distribución asintótica del mínimo global. El Teorema de Fisher y Tipper establece que esta distribución debe ser una de las tres distribuciones de Gumbel: tipo I, tipo II o tipo III. En particular, se trata de la distribución asintótica tipo III o de Weibull. De esta forma, se pueden obtener estimaciones puntuales e intervalos de confianza para el óptimo global, y reglas de parada para el método. El trabajo analiza varias metodologías para obtener las anteriores estimaciones e intervalos de confianza y compara experimentalmente las diferentes reglas de parada que, consecuentemente, se obtienen.

Usando un esquema Bayesiano, Betrò y Schoen [3] asumen una distribución a priori sobre los valores objetivos de los óptimos locales encontrados, y utilizan ésta para obtener reglas de parada.

Si el número de mínimos locales, κ, de la función objetivo fuese conocido, un criterio de parada obvio sería desarrollar búsquedas locales hasta encontrarlos todos. Sin embargo, este valor es desconocido. No obstante, el número de veces que aparece cada uno de los mínimos encontrados al aplicar las búsquedas locales suministra información sobre κ y sobre el tamaño de las correspondientes regiones de atracción. Boender y Rinnooy Kan [5] realizan un estudio detallado de estos parámetros siguiendo la metodología bayesiana. En ésta se supone que los parámetros son variables aleatorias para las que se asume una distribución a priori conocida que luego se modifica por las evidencias muestrales. Así, obtienen, inicialmente, dos reglas de parada: parar cuando se haya encontrado un número de mínimos locales distintos mayor o igual que el estimador bayesiano entero óptimo de κ; parar cuando el estimador del tamaño relativo de las regiones de atracción muestreadas sea suficientemente grande.

Las reglas obtenidas por Boender y Rinnooy Kan no tienen en cuenta el coste que supone desarrollar nuevas búsquedas locales. Una metodología alternativa para obtener reglas de parada consiste en suponer que cada vez que se finaliza un método multiarranque se incurre en dos pérdidas: una *pérdida de finalización*, que depende del coste que supone finalizar la búsqueda antes de encontrar el mínimo global, y una *pérdida de ejecución*, que depende del coste de realizar nuevas búsquedas locales. Boender y Zielinski [7] y Boender y Rinnooy Kan [4][5] consideran tres estructuras de pérdida según el esquema anterior y, para cada una de ellas, obtienen reglas de parada.

Moreno et al. [34] proponen una regla de parada para el método multiarranque basada en el estudio estadístico del número de iteraciones necesarias para encontrar el óptimo global. Los autores introducen dos variables aleatorias cuya

combinación proporciona el número de iteraciones necesarias hasta encontrar el óptimo global (variable ψ). Estas variables son: el número de soluciones iniciales generadas hasta que la correspondiente búsqueda local alcanza el óptimo global y el número de evaluaciones de la función objetivo hasta que la correspondiente búsqueda local alcanza un óptimo local. Si bien la distribución exacta de la variable que suministra el número de iteraciones necesarias para encontrar el óptimo global es difícil de obtener, si puede aproximarse apropiadamente usando la distribución normal. El criterio de parada propuesto consiste en finalizar la búsqueda después de e iteraciones siempre que la probabilidad de que ψ sea menor que e sea suficientemente grande. Por tanto, la regla de parada propuesta es, tras cada aplicación de una búsqueda local, verificar si la anterior condición es cierta o no. Si la respuesta es afirmativa, el algoritmo finaliza; en caso contrario, se selecciona una nueva solución desde la que aplicar una búsqueda local.

Hart [20] describe diversas reglas de parada secuenciales para la Búsqueda Aleatoria Pura que se basan en la estimación del óptimo global de una función. A continuación, las modifica y generaliza para otros algoritmos secuenciales, y describe como pueden usarse en un multiarranque. De la experiencia computacional desarrollada concluye que estas reglas de parada se comportan de forma similar a las reglas de parada bayesianas propuestas por [3][5]. Además, son, a juicio del autor, más sencillas y fáciles de usar.

4 Métodos multi-arranque para el problema de la Máxima Diversidad

El problema de seleccionar un subconjunto de elementos de diversidad máxima de un conjunto dado se conoce como Problema de la Máxima Diversidad o MDP (del inglés *Maximum Diversity Problem*). Este problema tiene multitud de aplicaciones prácticas entre las que destacan: tratamientos médicos, balanceo de sistemas ecológicos, políticas de inmigración o ingeniería genética entre otros [18]. El MDP ha sido estudiado por numerosos autores, entre los que destacan Kuo et al. [25] donde se describen cuatro formulaciones del problema, desde la primera más intuitiva a la última más eficiente, sirviendo también éstas para demostrar que el MDP es NP-difícil. En 1996 Ghosh [16] propone un método multiarranque y se demuestra la completitud del problema. Posteriormente, Glover y otros [18] proponen cuatro métodos heurísticos deterministas, dos de ellos constructivos y los otros dos destructivos. Silva y otros [39] presentan un algoritmo multi-arranque basado en la metodología GRASP. Concretamente describen tres métodos constructivos, denominados KLD, KLDv2 y MDI y dos métodos de mejora: LS, que es una adaptación del que propuso Ghosh, y SOMA, basado en una implementación VNS. Desde un punto de vista formal, el MDP se describe como un problema de optimización combinatoria que se peude formular como sigue:

sea $S = \{s_i : i \in N\}$ un conjunto de elementos donde $N = \{1, 2, \ldots, n\}$ es el conjunto de índices. Cada elemento del conjunto $s_i \in S$, se puede representar por un vector $si = (s_{i1}, s_{i2}, .., s_{ir})$. Sea d_{ij} la distancia entre dos elementos s_i y s_j y sea m (con $m < n$) el tamaño deseado del conjunto de máxima diversidad. En este contexto, la resolución del MDP consiste en encontrar un subconjunto Sel de m elementos de S ($Sel \subset S$ y $|Sel| = m$) de tal forma que se maximice la suma de las distancias entre los elementos seleccionados. Matemáticamente, el MDP se puede reescribir como un problema de decisión en los siguientes términos:

$$\max z = \sum_{i<j} d_{ij} x_i x_j$$

sujeto a:

$$\sum_{i=1}^{n} x_i = m$$

$$x_i \in \{0, 1\}\ i = 1, \ldots, n$$

donde $x_i = 1$ indica que el elemento s_i ha sido seleccionado.

Para abordar la resolución del MDP mediante un esquema multi-arranque, se proponen dos algoritmos constructivos, uno de ellos sin memoria y el otro con memoria. En las siguientes secciones se describe cada uno de estos algoritmos.

4.1 Multi-Arranque Sin Memoria (MASM)

El algoritmo Multi-Arranque Sin Memoria (MASM) propuesto en este trabajo consta de un procedimiento constructivo basado en GRASP y una búsqueda local tipo firts improvement. Esta propuesta se inspira en un heurístico propuesto por Glover y otros [18]. El procedimiento constructivo, en cada paso, añade un elemento de buena calidad (dado por una función tipo greedy) y se introduce en el conjunto Sel, de tal forma que en el conjunto $S \setminus Sel$ estarán los elementos no seleccionados. Inicialmente, el conjunto Sel está vacío; por lo tanto, todos los elementos podrían ser seleccionados. El algoritmo empezaría eligiendo aleatoriamente un elemento de S y lo introduciría en el conjunto Sel. Posteriormente, se calcula la distancia de todos los elementos no seleccionados $s_i \in S \setminus Sel$ al conjunto Sel como sigue:

$$d(s_i, X) = \sum_{j \in Sel} d(s_i, s_j) \tag{1}$$

que establecen una ordenación entre todos los elementos no seleccionados. Para seleccionar el siguiente elemento que se incluirá en el conjunto Sel, se construye una lista ordenada L donde estén todos los elementos $s_i \in S \setminus Sel$ con un porcentaje α de la máxima distancia. Matemáticamente, L se define como sigue:

$$L = \{s_i \in S \setminus Sel / d(s_i, s_c) \geq d_{min} + \alpha(d_{max} - d_{min})\} \tag{2}$$

donde

$$d_{max} = \max_{s_i \in S} d(s_i, Sel) \qquad d_{min} = \min_{s_i \in S} d(s_i, Sel)$$

El siguiente elemento que se introduce en el conjunto Sel se elige aleatoriamente de entre los elementos que hay en L, de tal forma que se asegura que tiene un porcentaje de calidad mínimo, fijado por α , y que no es una elección puramente *greedy*, que conduciría a un óptimo local. Este procedimiento se mantiene hasta haber seleccionado m elementos ($|Sel| = m$), de tal forma que en Sel se tendrá la solución al problema. Este constructivo se ejecutaría *niter* de tal forma que la media aritmética de las *niter* soluciones construidas será peor que si la solución se hubiese construido tomando el elemento con distancia máxima a los ya seleccionados, pero probablemente alguna de las *niter* soluciones mejore este valor.

Para tener un comportamiento reactivo del algoritmo, el parámetro α se inicializa a 0.5 y posteriormente se ajusta dinámicamente en función de la calidad de las soluciones construidas; es decir, si después de $niter/5$ iteraciones consecutivas, la solución con mejor valor no ha sido mejorada, entonces se incrementa α en 0.1 (hasta un máximo de 0.9).

Como método de mejora se presenta un procedimiento basado en una simplificación de la búsqueda local descrita en [16], que persigue el aumento de la eficiencia de la búsqueda local. El método propuesto se encuadra dentro de las búsquedas locales tipo *first improvement* que, como se describe en Laguna y otros [26], suele proporcionar mejores resultados que las estrategias tipo *best improvement*, obviamente en mucho menos tiempo. Para ello, se factoriza la aportación de cada elemento s_i en Sel; es decir, d_i para cada elemento $s_i \in Sel$ es la contribución a la función objetivo de cada elemento s_i:

$$d_i = \sum_{s_j \in Sel} d_{ij} = d(s_i, Sel) \tag{3}$$

ya que la función objetivo se define como

$$z = \frac{1}{2} \sum_{s_i \in Sel} d_i \tag{4}$$

Posteriormente, se selecciona el elemento s_{i^*} en Sel con menor contribución a la solución actual; es decir, el elemento $s_{i^*} \in Sel$ con menor valor de d_{i^*}, de tal forma que $s_{i^*} \in Sel$ se intercambia con el primer elemento $s_j \in S \setminus Sel$ que aumente el valor de la función objetivo. El procedimiento de búsqueda se mantiene mientras que haya mejora en la función objetivo, de tal forma que se extrae el elemento del conjunto Sel que menos aporte y se introduce otro de $S \setminus Sel$ que mejore el valor dicha función objetivo. Cuando no se obtenga una mejora, se pasa al segundo elemento que menos aporte y así sucesivamente. Este procedimiento se mantiene hasta que no se pueda conseguir ninguna mejora.

4.2 Multi-Arranque Con Memoria (MACM)

Como segundo algoritmo multi-arranque, denominado Multi-Arranque Con Memoria (MACM), en [11] se presenta un método que utiliza memoria tanto en la fase de construcción de soluciones como en la fase de mejora. Estas estrategias se describen dentro de la metodología Tabu Search [17].

El algoritmo constructivo se basa en una penalización por frecuencia en cada iteración a aquellos elementos que aparecieron en soluciones pasadas. El procedimiento también premia a aquellos elementos que aparecieron en soluciones pasadas de muy alta calidad. Para implementar dicho algoritmo se guarda en $freq[i]$ el número de veces que el elemento s_i ha sido seleccionado en construcciones previas. En max_freq se almacena el valor máximo de $freq[i]$ para todo i. Por otro lado, en $quality[i]$ se guarda el valor medio de las soluciones en las que ha participado el elemento s_i. Además, en max_q se almacena el máximo valor de $quality[i]$ para todo i. En estas condiciones, se modifica la evaluación del atractivo de cada elemento no seleccionado en la construcción actual de acuerdo a estas magnitudes para así favorecer la selección de elementos con baja frecuencia y alta calidad. Para ello, en vez de utilizar la distancia descrita en (3) entre un elemento y el conjunto de los elementos seleccionados, se utiliza la siguiente expresión:

$$d'(s_i, Sel) = ds_i, Sel - \beta range(Sel)\frac{freq[i]}{max\_freq} + \delta range(Sel)\frac{quality[i]}{max\_q}$$

con

$$range(Sel) = \max_{s_j \in S \setminus Sel} d(s_j, Sel) - \min_{s_j \in S \setminus Sel} d(s_j, Sel)$$

donde β y δ son parámetros del algoritmo que cuantifican la aportación de la penalización por frecuencia y la bonificación por calidad. Ambos se ajustan experimentalmente. La introducción del valor de $range(Sel)$ se hace para suavizar los cambios en la función de penalización.

Inicialmente, el conjunto Sel está vacío; por lo tanto, todos los elementos podrían ser seleccionados. El algoritmo empezaría seleccionando aleatoriamente un elemento de S y lo introduciría en el conjunto Sel. Posteriormente, calcula para cada elemento $s_i \in S \setminus Sel$ la distancia $d'(s_i, Sel)$, que en la primera construcción coincidirá con $d(s_i, Sel)$, ya que $freq[i] = quality[i] = 0$. Se elige aquél elemento que maximice dicha distancia:

$$i^* / d'(s_i^*, Sel) = \max_{s_i \in S}\{d'(s_i, Sel)\}$$

y se introduce en Sel, actualizando consecuentemente el vector de frecuencias. Este procedimiento se mantiene hasta haber seleccionado m elementos, de tal forma que en Sel se tendrá la solución al problema. Una vez que se tiene construida dicha solución, se actualiza el vector de calidad. El método tabú multi-

arranque ejecuta este procedimiento *niter* veces, de tal forma que en cada construcción las distancias entre un elemento al conjunto de los ya seleccionados se actualiza en función de su historia pasada.

Como método de mejora se utiliza una modificación del método de mejora propuesto anteriormente al que se le ha añadido memoria de corto plazo basado en intercambios entre Sel y $S \setminus Sel$. Una iteración de este algoritmo consiste en seleccionar aleatoriamente un elemento $s_i \in Sel$. La probabilidad de seleccionar dicho elemento es inversamente proporcional a su valor correspondiente de d_i. Ese elemento de Sel se sustituye por el primer elemento s_j de $S \setminus Sel$ que mejore el valor de la función objetivo. En el caso de que ningún elemento mejore el valor de la función objetivo, se selecciona aquél que menos la empeore, de tal forma que siempre se realiza un movimiento. Una vez ejecutado dicho movimiento, tanto s_i, como s_j adquieren estatus tabú durante $TabuTenure$ iteraciones. Por consiguiente, el elemento s_j no se podrá extraer del conjunto Sel (respectivamente, el elemento s_i del conjunto $S \setminus Sel$) durante este tiempo. El proceso de búsqueda tabú se mantiene hasta que se supere un número $MaxIter$ de iteraciones consecutivas sin que se mejore el mayor valor obtenido hasta el momento.

4.3 Resultados experimentales

Para ilustrar el comportamiento de los dos algoritmos multi-arranque resumidos en este trabajo y propuestos en [11], se presenta una comparativa con otros dos algoritmos previos. Concretamente, los algoritmos con los que se comparan MASM y MACM son el algoritmo constructivo D2, propuesto en Glover y otros [18] junto con el método de mejora descrito en [16] y el algoritmo KLDv2 con su correspondiente mejora, presentado en Silva y otros [39], que representan los mejores métodos para este problema. Todos los algoritmos fueron codificados en C y compilados con Borland Builder 5.0, optimizado para máxima velocidad. Los experimentos se ejecutaron en un Pentium IV a 3GHz con 1 GB RAM. Los algoritmos se probaron en tres conjuntos de ejemplos:

1. Silva: 20 matrices $n \times n$ con valores aleatorios enteros generados con una distribución uniforme de $[0, 9]$ con $n \in [100, 500]$ y $m \in [0.1n, 0.4n]$.
2. Glover: 20 matrices $n \times n$ en el que los valores de distancias entre cada par de puntos con coordenadas euclídeas se generan aleatoriamente en el plano $[0, 10]$. Para cada instancia, cada uno de estos n puntos tiene r coordenadas, con $r \in [2, 21]$.
3. Random: 20 matrices $n \times n$ con pesos reales generados con una distribución uniforme de $[0, 10]$ con $n = 2000$ y $m = 200$. Indicar que en la bibliografía consultada, estas son las instancias más grandes que se han resuelto.

En las tablas 2, 3 y 4 se comparan MASM, MACM, D2 + LS y KLDv2+LS. Estas tablas muestran para cada procedimiento el porcentaje medio de desviación

	$D_2 + LS$	$KLDv2 + LS$	$MASM$	$MACM$
Dev.	1.722%	1.079%	0.0377%	0.0130%
♯ Best	2	5	12	13
♯ Const.	5140.5	2663.6	925.4	864.1

Tabla 2: Métodos constructivos. Ejemplos tipo *Silva*

	$D_2 + LS$	$KLDv2 + LS$	$MASM$	$MACM$
Dev.	0.018%	0.006%	0.0000%	0.0000%
♯ Best	16	18	20	20
♯ Const.	2149.6	971.0	790.4	397.5

Tabla 3: Métodos constructivos. Ejemplos tipo *Glover*

con respecto a la mejor solución conocida (en cada experimento, ya que no se conocen los valores óptimos), el número de veces que el algoritmo encuentra la mejor solución y el número de construcciones y mejora que hace el algoritmo en 10 segundos (criterio de parada). La conclusión que se puede obtener de estas tablas es que los métodos multi-arranque propuestos mejoran sustancialmente los algoritmos previos tanto en desviación con respecto al mejor valor conocido como al número de veces que encuentra ese valor. Además, en la experimentación presentada también se pude concluir que el uso de memoria, al menos para este problema y estos ejemplos, conduce a mejores resultados. Indicar que en el caso de los ejemplos de tipo Glover, los algoritmos estudiados obtienen valores muy similares, de lo que se deduce que éstos son los más sencillos; como consecuencia, permiten discernir poco sobre la calidad de cada algoritmo. En el otro extremo estarían los ejemplos de tipo Random, donde se observa claramente que los métodos multi-arranque propuestos mejoran sustancialmente a los algoritmos previos.

	$D_2 + LS$	$KLDv2 + LS$	$MASM$	$MACM$
Dev.	1.270%	1.219%	0.204%	0.099%
♯ Best	0	0	7	15
♯ Const.	128.1	3.5	12.0	14.8

Tabla 4: Métodos constructivos. Ejemplos tipo *Random*

5 Bibliografía

[1] D.H. Ackley An Empirical Study of Bit Vector Function Optimization. En *Genetic Algorithms and Simulated Annealing*, pp. 170-204 Ed. Davis, Morgan Kaufmann Publishers (1987)

[2] S. Baluja *An Empirical Comparison of 7 Iterative Function Optimization Heuristics* Technical Report CMU–CS–95–193, School of Computer Science, Carnegie Mellon University (1995)

[3] B. Betrò, F. Schoen Sequential Stopping Rules for the Multistart Algorithm in Global Optimization *Mathematical Programming* vol. 38 pp. 271-286 (1987)

[4] C.G.E. Boender, A.H.G. Rinnooy Kan A Bayesian Analysis of the Number of Cells of a Multinomial Distribution *The Statistician* vol. 32 pp. 240-248 (1983)

[5] C.G.E. Boender, A.H.G. Rinnooy Kan Bayesian Stopping Rules for Multistart Global Optimization Methods *Mathematical Programming* vol. 37 pp. 59-80 (1987)

[6] C.G.E. Boender, A.H.G. Rinnooy Kan, L. Stougie, G.T. Timmer. A Stochastic Method for Global Optimization. *Mathematical Programming* vol. 22 pp. 125-140 (1982)

[7] C.G.E. Boender, R. Zielinski A Sequential Bayesian Approach to Esimating the Dimension of Multinomial Distribution. En R. Zielinski, editor, *Sequential Methods in Statistics*. Banach Center Publications vol. 16, PWN-Polish Scientif Publishers, Warsaw (1982)

[8] K.D. Boese, A.B. Kahng, S. Muddu A New Adaptive Multistart Technique for Combinatorial Global Optimisation *Operations Research Letters* vol. 16 pp. 103-113 (1994)

[9] R. H. Byrd, C.L. Dert, A.H.G. Rinnooy Kan, R. B. Schnabel Concurrent Stochastic Methods for Global Optimization *Mathematical Programming* vol. 46 pp. 1-29 (1990)

[10] E. Cuthill, J. McKee Reducing the Bandwidth of Sparse Symmetric Matrices. En *Proc. ACM National Conference, Association for Computing Machinery*, New York, pp. 157-172 (1969)

[11] A. Duarte, R. Martí (2006), Tabu Search and GRASP for the Maximum Diversity Problem *European Journal of Operational Research* (2006) En imprenta.

[12] T.A. Feo, M.G.C. Resende A Probabilistic Heuristic for A Computationally Difficult Set Covering Problem. *Operations Research Letters* vol. 8 pp. 67-71 (1989)

[13] T.A. Feo, M.G.C. Resende Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization* vol. 6 pp. 109-133 (1995)

[14] C. Fleurent, F. Glover Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. *INFORMS Journal on Computing* vol. 11 pp. 198-204 (1999)

[15] Fylstra Frontline Systems. http: www.frontsys.com

[16] J.B. Ghosh Computational aspects of the maximum diversity problem. *Operations Research Letters* vol. 19 pp. 175-181 (1996)

[17] F. Glover, M. Laguna *Tabu Search*. Kluwer Academic Publisher, (1997)

[18] F. Glover, C.C. Kuo, K.S. Dhir Heuristic algorithms for the maximum diversity problem. *Journal of Information and Optimization Sciences* vol 19 (1) pp. 109-132 (1998)

[19] L.W. Hagen, A.B. Kahng A.B. Combining Problem Reduction and Adaptive Multistart: A New Technique for Superior Iterative Partitioning *IEEE Transactions on CAD* vol. 16 pp. 709-717 (1997)

[20] W.W. Hart Sequential Stopping Rules for Random Optimization Methods with Applications to Multistart Local Search *Siam Journal on Optimization* vol. 9 pp. 270-290 (1998)

[21] J.P. Hart, A.W. Shogan Semi-greedy Heuristics: An Empirical Study, *Operations Research Letters* vol. 6 pp. 107-114 (1987)

[22] F.J. Hickernell, Y. Yuan A Simple Multistart Algorithm for Global Optimization, *OR Transactions* vol. 1 (1997)

[23] C.R. Houck, J.A. Joines, M.G. Kays Comparison of Genetic Algorithms, Random Restart and Two-Opt Switching for Solving Large Location-Allocation Problems, *Computers Operations Resarch*, vol. 23 pp. 587-596 (1996)

[24] X. Hu, R. Shonkwiler, M.C. Spruill *Random Restarts in Global Optimization* Georgia Institute of technology, Atlanta. (1994)

[25] C.C. Kuo, F. Glover, K.S. Dhir Analyzing and modeling the maximum diversity problem by zero-one programming. *Decision Sciences* vol. 24 (6) pp. 1171-1185 (1993)

[26] M. Laguna, R. Martí, V. Campos Intensification and diversification with elite tabu search solutions for the linear ordering problem *Computers and Operations Research* vol. 26 pp. 1217-1230 (1999)

[27] M. Locatelli, F. Schoen Random Linkage: A Family of Acceptance-Rejection Algorithms for Global Optimization *Mathematical Programming* vol. 85 pp. 379-396 (1999)

[28] M. Los, C. Lardinois Combinatorial Programming, Statistical Optimization and the Optimal Transportation Network Problem. *Transportation Research* vol. 2 pp. 89-124 (1982)

[29] R. Martí Multistart Methods, En *Handbook on MetaHeuristics*, pp. 355-368 Eds. F. Glover y G . Kochenberger, Kluwer (2003)

[30] R. Martí, M. Laguna, F. Glover, V. Campos Reducing the Bandwidth of a Sparse Matrix with Tabu Search, *European Journal of Operational Research* vol. 135 pp. 211-220 (2001)

[31] R. Martí, J. Marcos Moreno-Vega Métodos Multiarranque. *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial* vol. 19 pp. 49-60 (2003)

[32] D.Q. Mayne, C.C. Meewella A Non-Clustering Multistart Algorithm for Global Optimization. En *Analysis and Optimization of Systems*, Ed. Bensoussan and Lions, Lecture Notes in Control and Information Sciences, vol. 111, Springer Verlag (1988)

[33] M.B. Melián-Batista, J. A. Moreno, J.M. Moreno-Vega A Multistart Clustering Tecnhique for Combinatorial Optimization, en *Proceedings of the International Conference on Modelling and Simulation*, pp. 839-845 Eds. R. Berriel, V. Hernández, R. Montenegro, J. Rocha, Universidad de Las Palmas de Gran Canaria, (2000)

[34] J.A. Moreno, N. Mladenovic, J. M. Moreno-Vega *An Statistical Analysis of Strategies for Multistart Heuristic Searches for p-Facility Location-Allocation Problems* Eighth Meeting of the EWG on Locational Analysis, Lambrecht, Germany (1995)

[35] E. Piñana, I. Plana, V. Campos, R. Martí GRASP and Path Relinking for the Matrix Bandwidth Minimization, *European Journal of Operational Research*, por aparecer (2003)

[36] M. Prais, C.C. Ribeiro Reactive GRASP: An Application to a Matrix Decomposition Problem inf TDMA Traffic Assignment, *Informs Journal on Computing* vol. 12 pp. 164-176 (2000)

[37] A.H.G. Rinnooy Kan, G.T. Timmer Stochastic Global Optimization Methods. Part II: Multi Level Methods *Mathematical Programming* vol. 39 pp. 57-78 (1987)

[38] F. Schoen Two Phase Methods for Global Optimization. En *Handbook of Global Optimization 2: Heuristic Approaches*, pp. 151-178 Eds. P. Pardalos y E. Romeijn, Kluwer Academic Publishers (2002)

[39] G.C. Silva, L.S. Ochi, S.L. Martins Experimental comparison of greedy randomized adaptive search procedures for the maximum diversity problem, *Lecture Notes in Computer Science* vol. 3059 pp. 498-512, Springer, (2004)

[40] F. Solis, R. Wets. Minimization by Random Search Techniques. *Math. of Operations Research* vol. 6 pp. 19-30. (1981)

[41] Z. Ugray, L. Lasdon, J. Plummer, F. Glover, J. Kelly, R. Martí A Multistart Scatter Search Heuristic for Smooth NLP and MINLP Problems, enviado a *Informs Journal on Computing* (2001)

[42] N.L.J. Ulder, E.H.L. Aarts, H.J. Bandelt, P.J.M. Van Laarhoven, E. Pesch Genetic Local Search Algorithms for the Traveling Salesman Problem, *Parallel problem solving from nature*, Eds. Schwefel y Männer, Springer Verlag, pp. 109-116 (1990)

[43] M. Wattenberg, A. Juels *Stochastic Hillclimbing as a Baseline Method for Evaluationg Genetic Algorithms* University of California - Berkeley, CSD94-834 (1994)

Tercera parte

Aplicaciones

GRASP y Tabu Search para problemas de corte bidimensional no-guillotina*

R. Alvarez-Valdes[a], **F. Parreño**[b], **J.M. Tamarit**[a]

[a] Universidad de Valencia,
Departamento de Estadística e Investigación Operativa
[b] Universidad de Castilla-La Mancha,
Departamento de Informática

1 Introducción

El problema de corte bidimensional no-guillotina que estudiamos en este trabajo consiste en cortar un conjunto de pequeñas piezas rectangulares de un tablero rectangular grande de manera que se maximice el beneficio de las piezas cortadas. El problema aparece en muchos procesos productivos, en las industrias del textil, papel, acero, madera o cristal, en las que los tableros grandes obtenidos en la fase de producción se han de cortar en piezas más pequeñas para atender las demandas de los clientes, o cuando cajas rectangulares se han de colocar en containers y sólo dos dimensiones son relevantes. Patrones de corte eficientes reducen las pérdidas de material y estrategias de empaquetado eficientes mejoran la utilización del espacio y reducen los costes de transporte.

En este trabajo no imponemos a los patrones de corte la restricción de utilizar cortes guillotina, cortes que dividen completamente un rectángulo en dos subrectángulos. Esta restricción, que es muy común en algunas industrias, como las de la madera o el vidrio, aparece en muchos trabajos, pero no es necesaria

*Este trabajo ha sido financiado parcialmente por el Proyecto PBC-02-002, Consejería de Ciencia y Tecnología, JCCM, y el Ministerio de Educación y Ciencia DPI2005-04796.

cuando se utilizan nuevas tecnologías de corte y obviamente no se necesita en problemas de empaquetamiento. Sin esta restricción, se pueden obtener mejores soluciones, pero la complejidad del problema aumenta considerablemente. Los algoritmos exactos existentes sólo pueden resolver problemas de pequeño tamaño y por tanto es necesario el uso de algoritmos heurísticos. Nosotros proponemos primero un algoritmo constructivo y, a partir de él, un algoritmo GRASP y un algoritmo Tabu Search. Los resultados de un extenso estudio computacional muestran la eficiencia de dichos algoritmos.

2 Descripción del problema

El problema de corte bidimensional no-guillotina puede describirse de la forma siguiente. Sea $R = (L, W)$ el rectángulo grande, con longitud L y anchura W. Cada pieza i tiene dimensiones (l_i, w_i), y valor v_i, $i = 1, \ldots, m$. Las piezas tienen orientación fija y han de ser cortadas con sus lados paralelos a los lados del rectángulo (cortes ortogonales). El problema consiste en cortar el rectángulo R en x_i copias de cada pieza i, de manera que $0 \leq P_i \leq x_i \leq Q_i$, y el valor total de las piezas cortadas, $\sum_i v_i x_i$, sea máximo. Denotaremos $M = \sum_i Q_i$ el número máximo de piezas que se pueden cortar. El problema se clasifica como 2D-SLOPP (*2-Dimensional Single Large Object Placement Problem*) en la clasificación propuesta por Wäscher et al. (2006).

Según los valores de P_i y Q_i, podemos distinguir tres tipos de problemas:

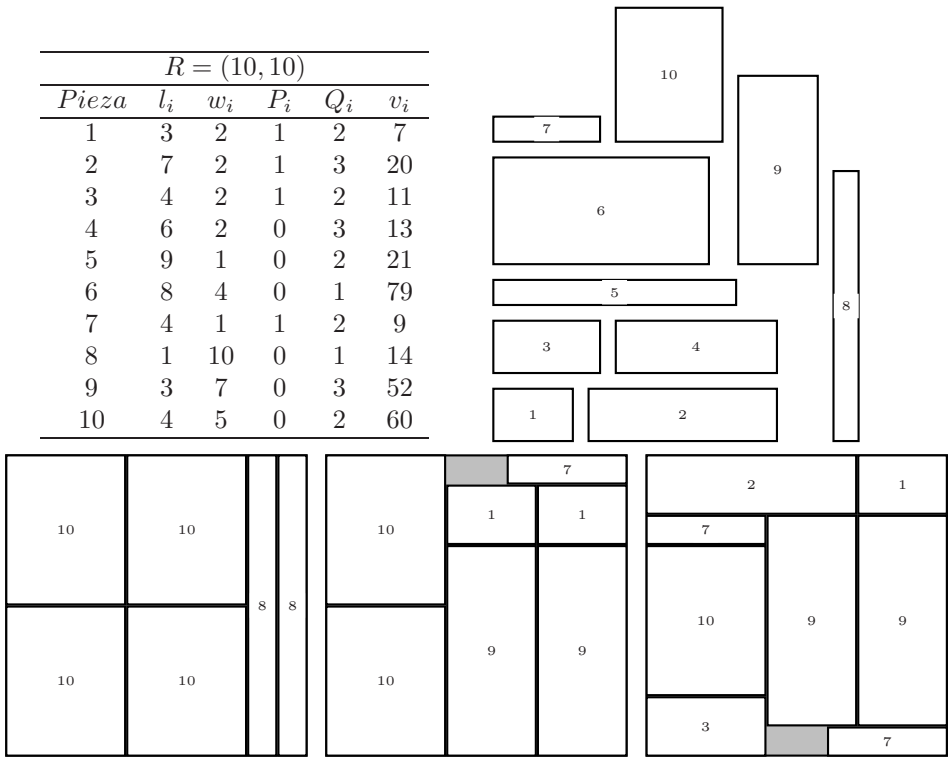
1. *No restringido:* $\forall i,\ P_i = 0$, $Q_i = \lfloor L * W / l_i * w_i \rfloor$ (cota trivial).
2. *Restringido:* $\forall i,\ P_i = 0$; $\exists i, Q_i < \lfloor L * W / l_i * w_i \rfloor$
3. *Doblemente restringido:* $\exists\, i,\ P_i > 0$; $\exists\, j,\ Q_j < \lfloor L * W / l_j * w_j \rfloor$

En la Figura 1 vemos un ejemplo, con un rectángulo $R = (10, 10)$, y $m = 10$ piezas que cortar. La primera solución, Figura 1(b), es óptima para el problema irrestringido, mientras que la segunda solución, Figura 1(c), corresponde al caso restringido y la tercera, Figura 1(d), al doblemente restringido, con algunos $P_i \neq 0$.

Algunos autores han estudiado el problema irrestringido: Tsai et al. (1988), Arenales y Morabito (1995), Healy et al. (1999). Sin embargo, el problema restringido es más interesante para las aplicaciones y ha recibido mucha mayor atención. Métodos exactos han sido propuestos por Beasley (1985), Scheithauer y Terno (1993), Hadjiconstantinou y Christofides (1995), Fekete y Schepers (2004) y Caprara y Monaci (2004).

Una sencilla cota superior para el problema se obtiene resolviendo el siguiente problema mochila acotado, en el que la variable x_i representa el número de piezas

$R = (10, 10)$					
Pieza	l_i	w_i	P_i	Q_i	v_i
1	3	2	1	2	7
2	7	2	1	3	20
3	4	2	1	2	11
4	6	2	0	3	13
5	9	1	0	2	21
6	8	4	0	1	79
7	4	1	1	2	9
8	1	10	0	1	14
9	3	7	0	3	52
10	4	5	0	2	60



(b) No restringido Opt: 268 (c) Restringido Opt: 247 (d) Doblemente restr. Opt: 220

Figura 1: *Problema 3 de Beasley (1985)*

de tipo i que se cortan por encima de su cota inferior P_i:

$$Max \quad \sum_{i=1}^{m} v_i x_i + \sum_{i=1}^{m} v_i P_i \tag{1}$$

$$s.t.: \quad \sum_{i=1}^{m} (l_i w_i) x_i \leq LW - \sum_{i=1}^{m} P_i (l_i w_i) \tag{2}$$

$$x_i \leq Q_i - P_i, \;\; i = 1, \ldots, m \tag{3}$$

$$x_i \geq 0, \quad integer, \;\; i = 1, \ldots, m. \tag{4}$$

Otras cotas, aparte de las incluídas en los métodos exactos mencionados, han sido propuestas por Scheithauer (1999) y Amaral y Letchford (2003).

Recientemente han ido apareciendo diversos algoritmos heurísticos. Wu et al. (2002) proponen un algoritmo constructivo. Lai y Chan (1997) y Leung el al. (2001, 2003) utilizan templado simulado y algoritmos genéticos. Beasley

(2004) desarrolla un algoritmo genético basado en una formulación no lineal del problema. Presenta además un estudio computacional muy completo sobre un conjunto de problemas test estandard y sobre un nuevo conjunto de problemas grandes generado aleatoriamente.

En este artículo describimos nuestros trabajos en el desarrollo de dos tipos de algoritmos, GRASP y Tabu Search, e incluímos los resultados obtenidos sobre cuatro conjuntos de problemas test: los 21 problemas de la literatura utilizados por Beasley (2004); los 630 problemas grandes, generados aleatoriamente por Beasley (2004); 10 problemas utilizados por Leung et al. (2003), y los 21 problemas utilizados por Hopper y Turton (2001). Este último conjunto fue inicialmente diseñado para otro problema de empaquetamiento bidimensional y ha sido adaptado a nuestro problema para probar nuestros algoritmos sobre problemas difíciles en los que la solución óptima no contiene pérdidas. Más detalles sobre nuestros algoritmos pueden encontrarse en Alvarez-Valdes et al. (2005).

3 Un algoritmo constructivo

Seguimos un procedimiento iterativo en el que combinamos dos elementos: una lista $\mathcal{P}$ de piezas por cortar, inicialmente la lista completa de piezas, y una lista $\mathcal{L}$ de rectángulos vacíos en los que se puede cortar una pieza, que contiene inicialmente el rectángulo $R = (L, W)$. En cada paso se elige un rectángulo de $\mathcal{L}$, y de las piezas de $\mathcal{P}$ que caben en él, se elige la pieza a cortar. Al cortar la pieza se pueden producir nuevos rectángulos, que se añaden a $\mathcal{L}$ y el proceso continúa hasta que $\mathcal{L} = \emptyset$ o ninguna de las piezas restantes cabe en ninguno de los rectángulos de $\mathcal{L}$.

Paso 0. Inicialización:

$\mathcal{L} = \{R\}$, el conjunto de rectángulos vacíos.
$\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$, el conjunto de piezas pendientes de cortar.
El conjunto $\mathcal{P}$ se ordena inicialmente siguiendo 3 criterios: Ordenar por $P_i * l_i * w_i$ no creciente, dando prioridad a las piezas que se han de cortar obligatoriamente. Si se da un empate (por ejemplo, si $P_i = 0, \forall i$), ordenar por $v_i/(l_i * w_i)$ no creciente. Si hay empates (por ejemplo, si $v_i = l_i * w_i, \forall i$), ordenar por $l_i * w_i$ no creciente.

$\mathcal{B} = \emptyset$, el conjunto de piezas ya cortadas. Las piezas del mismo tipo pueden aparecer agrupadas en *bloques* rectangulares.

Paso 1. Elección del rectángulo:

Tomar R^*, el menor rectángulo de $\mathcal{L}$ en el que cabe una pieza $p_i \in \mathcal{P}$.
Si tal R^* no existe, parar.
En otro caso, ir al Paso 2.

Paso 2. Elección de la pieza:

Elegir una pieza p_i y una cantidad $n_i \leq Q_i$, formando el bloque B^* para cortarlo de R^*.
Se elige la pieza i que produzca el mayor aumento de la función objetivo.
El bloque B^* se corta en la esquina de R^* más próxima a una esquina del rectángulo R inicial.
Actualizar $\mathcal{P}$, $\mathcal{B}$ y Q_i que indica el número de piezas de tipo i que quedan por cortar.
Mover el bloque B^* hacia la esquina más próxima del rectángulo R.

Paso 3. Actualización de $\mathcal{L}$:

Añadir a $\mathcal{L}$ los posibles rectángulos producidos al cortar B^* de R^*.
Tener en cuenta los posibles cambios en $\mathcal{L}$ al mover el bloque B^*.
Fusionar rectángulos para favorecer el corte de nuevas piezas de $\mathcal{P}$.
Volver al Paso 1.

Aunque a lo largo del algoritmo nosotros mantenemos una lista de rectángulos vacíos $\mathcal{L}$, en realidad tenemos un espacio vacío poligonal irregular en el que las piezas pendientes podrían tener cabida. Una manera de adaptar nuestra lista $\mathcal{L}$ a la flexibilidad del corte no-guillotina es fusionar algunos de los rectángulos de la lista, produciendo nuevos rectángulos en los que las piezas pendientes puedan caber mejor.

Cuando fusionamos 2 rectángulos, pueden aparecer, como máximo, 3 nuevos rectángulos, típicamente un rectángulo *grande* y 2 *pequeños* (ver Figura 2). Entre las diversas alternativas de fusión, tratamos de seleccionar la mejor, es decir, aquélla en la que es posible cortar las piezas mejor situadas en la lista ordenada $\mathcal{P}$. Con este objetivo en mente, imponemos las siguientes condiciones:

1. Si el orden de la mejor pieza que cabe en el rectángulo *grande* es estrictamente menor que el orden de las piezas en los rectángulos originales, los fusionamos.

2. Si el orden de la mejor pieza que cabe en el rectángulo *grande* es igual que el orden de las piezas en los rectángulos originales, los fusionamos si el área del rectángulo grande es mayor que el área de cada uno de los rectángulos originales.

3. Si el orden de la mejor pieza que cabe en el rectángulo *grande* es estrictamente mayor que el orden de las piezas en los rectángulos originales, no los fusionamos.

En la Figura 2 vemos varios casos posibles. En la Figura 2(a) los rectángulos originales siempre se fusionarán. El nuevo rectángulo es mayor que ambos y todas las piezas que cabían en los originales cabrán en él. En la Figura 2(b) los nuevos

rectángulos no son mayores que los originales. Éstos se fusionarán sólo si el nuevo rectángulo central permite cortar una pieza de menor orden que las que cabían en los originales. En la Figura 2(c) uno de los nuevos rectángulos es mayor que los originales y por tanto éstos se fusionarán, a menos que la mejor pieza que cabía en el rectángulo original vertical no quepa en los nuevos rectángulos.

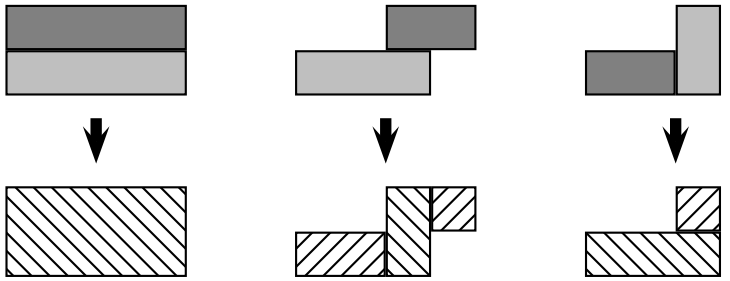


Figura 2: *Fusionando 2 rectángulos vacíos*

Al final de proceso constructivo, una solución está compuesta por una lista de bloques $\mathcal{B}$, y una lista de rectángulos vacíos $\mathcal{L}$, con valor total $\sum_i v_i x_i$.

4 Un algoritmo GRASP

GRASP (Greedy Randomized Adaptive Search Procedure) fue desarrollado por Feo y Resende (1989) para resolver problemas combinatorios difíciles. Para una introducción actualizada, consultar Resende y Ribeiro (2003). GRASP es un procedimiento iterativo que combina una fase constructiva y una fase de mejora. En la fase constructiva se construye paso a paso una solución posible, añadiendo elementos a una solución parcial. El elemento a añadir en cada paso se elige mediante una función *greedy* que se adapta dinámicamente a lo largo del proceso. Sin embargo, esta elección no es determinista, sino sujeta a un proceso de aleatorización. De esta forma, al repetir el proceso se pueden obtener soluciones diferentes. Al acabar cada fase constructiva, una fase de mejora, que suele consistir en una búsqueda local, intenta sustituir algunos elementos que forman parte de la solución debido a la aleatorización por otros que mejoren la calidad de la solución.

4.1 Fase constructiva

En nuestro algoritmo GRASP, la fase constructiva corresponde al algoritmo constructivo descrito en la sección anterior, introduciendo procedimientos de aleatorización al elegir la pieza a cortar. Sea s_i el valor de la pieza (o bloque de piezas) i y $s_{max} = max\{s_i | i \in \mathcal{P}\}$, y sea δ un parámetro a determinar ($0 < \delta < 1$). Hemos considerado tres alternativas:

1. Elegir la pieza i al azar en el conjunto $S = \{j \mid s_j \geq \delta s_{max}\}$
   (S suele denominarse *Conjunto restringido de Candidatos*).
2. Elegir la pieza i al azar entre el mejor $100\,(1-\delta)\%$ de las piezas.
3. Elegir la pieza i en todo el conjunto $\mathcal{P}$, pero con probabilidades proporcionales a sus valores s_i ($p_i = s_i / \Sigma s_j$)

4.2 Elección del parámetro δ

Un estudio preliminar mostró que no existía ningún valor de δ que produjera siempre los mejores resultados. Por tanto, consideramos varias alternativas en las que el valor de δ variaba aleatoria o sistemáticamente a lo largo de las iteraciones. Estas estrategias fueron:

1. En cada iteración, elegir δ al azar en el intervalo [0.4, 0.9]
2. En cada iteración, elegir δ al azar en el intervalo [0.25, 0.75]
3. En cada iteración δ toma por turno uno de los siguientes 6 valores:
   0.4, 0.5, 0.6, 0.7, 0.8, 0.9.
4. $\delta = 0.75$
5. *GRASP Reactivo*
   En el GRASP Reactivo, propuesto por Prais y Ribeiro (2000), δ se toma inicialmente al azar de un conjunto dado de valores discretos, pero transcurrido un cierto número de iteraciones se analiza la calidad relativa de las soluciones obtenidas con cada valor de δ y se aumentan las probabilidades de los valores que producen mejores soluciones.

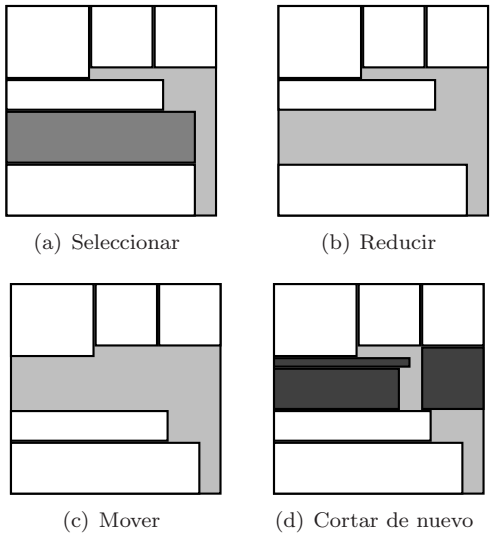
4.3 Fase de mejora

Cada solución construida en la fase anterior es el punto de partida para una búsqueda local en la que tratamos de mejorarla. Hemos estudiado tres alternativas:

I) Tomamos un bloque adyacente a un rectángulo vacío y consideramos reducirlo o eliminarlo completamente. Los demás bloques se desplazan hacia las esquinas, los rectángulos vacíos resultantes se fusionan y la lista actualizada $\mathcal{L}$ se vuelve a cortar utilizando el algoritmo constructivo (Figura 3). Si la solución obtenida mejora la inicial, se realiza el movimiento y se pasa a estudiar otro bloque. Sólo se considera la reducción de un bloque cuando ello no viola las cotas inferiores P_i en el número de piezas a cortar.

II) El segundo procedimiento es una simplificación del método I en el que los bloques no se desplazan hacia las esquinas y los rectángulos vacíos sólo se fusionan con los ya existentes (Figura 4).

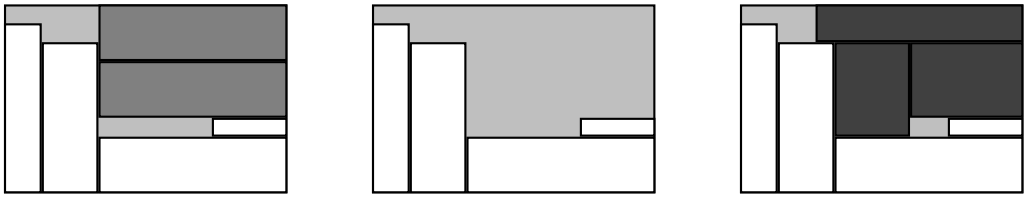
III) El tercer método consiste en eliminar el último $k\%$ de los bloques incorporados a la solución (por ejemplo, el último 10%) y volver a cortar el espacio vacío resultante con el algoritmo constructivo determinista, como proponen Beltran et al. (2002). Cuando estos últimos bloques han sido eliminados, los bloques restantes se desplazan hacia las esquinas y los rectángulos vacíos resultantes se fusionan, antes de aplicar el procedimiento constructivo (ver Figura 5, en la que los números en las piezas indican el orden en el que fueron incluidas en el proceso de corte).



(a) Seleccionar (b) Reducir

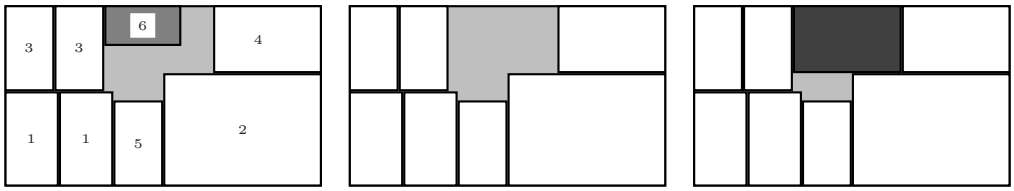
(c) Mover (d) Cortar de nuevo

Figura 3: *Método de mejora I. Problema 8, Tabla 1*



(a) Seleccionar (b) Reducir (c) Cortar de nuevo

Figura 4: *Método de mejora II. Problema 6, Tabla 1*



(a) Seleccionar (b) Eliminar (c) Cortar de nuevo

Figura 5: *Método de mejora III. Problema 15, Tabla 1*

5 Algoritmo Tabu Search

Tabu Search es actualmente un metaheurístico bien conocido (para una introducción, consultar el libro de Glover y Laguna (1997). Los elementos básicos del algoritmo se describen en los apartados siguientes.

5.1 Definición de movimientos

El espacio de soluciones en el que nos movemos está compuesto únicamente por las soluciones posibles. En este espacio definimos varios movimientos para ir de una solución a otra. La solución inicial se obtiene aplicando el algoritmo constructivo de la seccción 3.

Distinguimos dos tipos de movimientos: reducción de bloques e inserción de bloques. En la reducción de bloques, se reduce el tamaño de un bloque existente eliminando algunas de sus filas o columnas. En la inserción de bloques, un nuevo bloque se añade a la solución. En ambos casos, presentamos priemro un esquema del procedimiento y luego un ejemplo detallado.

- ***Reducción de bloques***

  Paso 0. Inicialización:

$\mathcal{B}$ = lista de bloques de la solución actual
$\mathcal{L}$ = lista de rectángulos vacíos

Paso 1. Elección del bloque a reducir

Tomar B, uno de los bloques de $\mathcal{B}$, con k columnas y l filas de piezas p_i.

Seleccionar el número de r de columnas (filas) a eliminar,

$$1 \leq r \leq k \quad (1 \leq r \leq l),$$

manteniendo el número de piezas en la solución $x_i \geq P_i$.

Si $P_i = 0$, el bloque puede desaparecer completamente.

El nuevo rectángulo vacío se añade $\mathcal{L}$.

Paso 2. Mover los bloques restantes hacia sus esquinas más próximas:

La lista de rectángulos vacíos $\mathcal{L}$ se actualiza adecuadamente.

Paso 3. Volver a cortar nuevos bloques en los rectángulos vacíos:

Aplicar el algoritmo constructivo de la Sección 3,

El algoritmo parte de las listas actualizadas $\mathcal{L}$ y $\mathcal{B}$, y $\mathcal{P}$ contiene las piezas pendientes de cortar. Antes de aplicar el procedimiento constructivo, se estudia las posibles fusiones de rectángulos de $\mathcal{L}$, para adaptarse lo mejor posible a las piezas de $\mathcal{P}$.

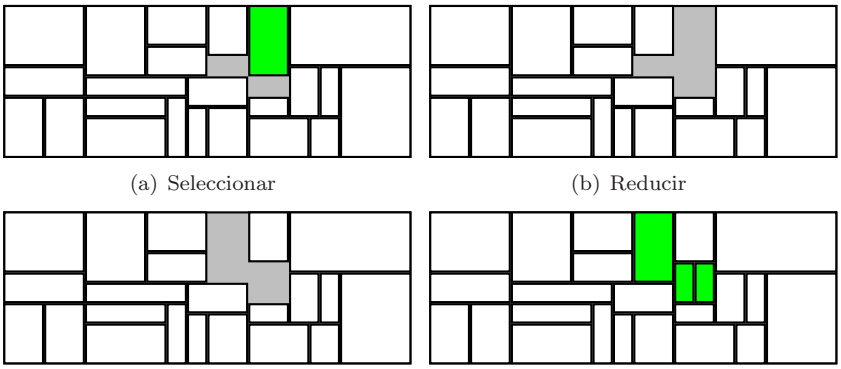
Al seleccionar la pieza a cortar, la pieza eliminada en el Paso 1 no se considera hasta que otra pieza haya sido añadida a la solución.

Paso 4. Fusionar los bloques con la misma estructura:

Fusionamos dos bloques de la misma pieza si son adyacentes (o uno de ellos puede desplazarse para ser adyacente con el otro) y en el lado adyacente tienen la misma dimensión.

En la Figura 6 vemos un ejemplo de movimiento de reducción sobre un problema propuesto por Jakobs (1996) y utilizado posteriormente por Leung et al. (2003). Los rectángulos se denotan (x_1, y_1, x_2, y_2) donde (x_1, y_1) son las coordenadas del vértice inferior izquierdo y (x_2, y_2) las del vértice superior derecho. El rectángulo inicial es $R = (0, 0, 120, 45)$, con $m = 22$ tipos de piezas y un total de $M = 25$ piezas que utilizan todo el rectángulo, sin dejar zonas de pérdida. La Figura 6(a) muestra una solución con 23 piezas, en la que no caben dos piezas (6x12). El conjunto $\mathcal{L}$ está compuesto por $R_1 = (60, 24, 72, 30)$ y $R_2 = (72, 18, 84, 24)$ (en gris claro). En el Paso 1, se selecciona un bloque compuesto por una pieza (12x21) (en gris oscuro). Su reducción supone su desaparición completa de la solución, creando un nuevo rectángulo vacío $R_3 = (72, 24, 84, 45)$ que se añade a $\mathcal{L}$ (Figura 6(b)).

En el Paso 2, el bloque compuesto por una pieza (12x15) se desplaza hacia la esquina superior derecha. Por tanto, $\mathcal{L} = \{R_1, R_2, R_4, R_5\}$, donde $R_4 = (60, 30, 72, 45)$ y $R_5 = (72, 24, 84, 30)$ (Figura 6(c)). En el Paso 3 el procedimiento constructivo vuelve a cortar los rectángulos vacíos. Primero, R_1 y R_4 se fusionan, formando $R_6 = (60, 24, 72, 45)$, y lo mismo sucede con R_2 y R_5, que forman $R_7 = (72, 18, 84, 30)$. Entonces, se selecciona R_7 y las dos piezas (6x12) se cortan en él, llenándolo completamente. Finalmente, se toma R_6 y se corta en él la pieza eliminada inicialmente. La solución final, que es óptima, aparece en la Figura 6(d).



(a) Seleccionar (b) Reducir

(c) Mover hacia la esquina (d) Volver a cortar

Figura 6: *Reducción de bloques. Problema 3 de Jakobs (1996)*

- ***Inserción de bloques***

Paso 0. Inicialización:

$\mathcal{B}$ = lista de bloques
$\mathcal{L}$ = lista de rectángulos vacíos

Paso 1. Elegir el bloque a insertar

Tomar p_i, una pieza para la que $x_i < Q_i$, y considerar un bloque de esas piezas con k columnas y l filas $(k * l \leq Q_i - x_i)$.

Paso 2. Seleccionar la posición en la que insertar el nuevo bloque

Paso 3. Eliminar las piezas de la solución que se solapan con el nuevo bloque

Actualizar $\mathcal{B}$ (algunos de los bloques originales se reducen o eliminan)

Actualizar $\mathcal{L}$ (pueden aparecer nuevos rectángulos vacíos).

Paso 4. Volver a cortar nuevos bloques en los rectángulos vacíos

Paso 5. Fusionar los bloques con la misma estructura:

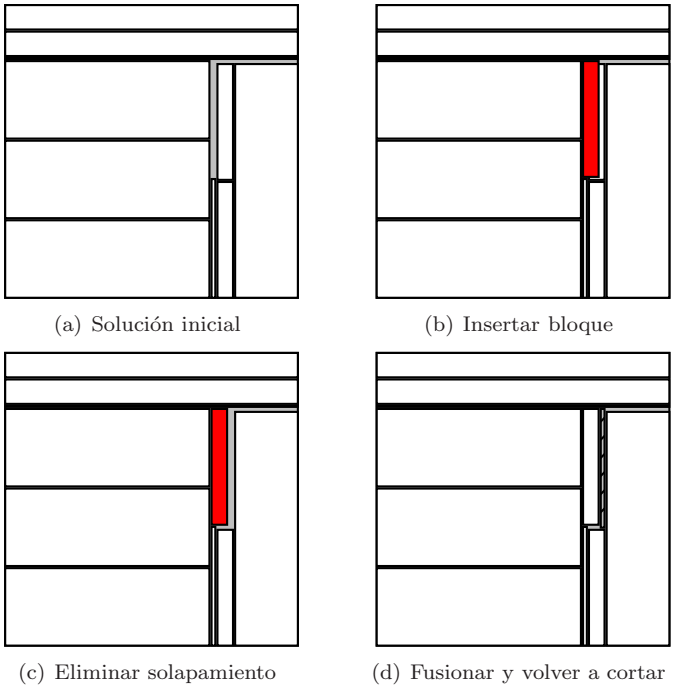
Los Pasos 4 y 5 son los mismos del procedimiento de reducción de bloques.

En la Figura 7 vemos un ejemplo de movimiento de inserción sobre un problema propuesto por Fekete y Schepers (2004) y utilizado posteriormente por Beasley (2004). El rectángulo inicial es $R = (0, 0, 100, 100)$ y se han de cortar $m = 15$ tipos de piezas, con un total de $M = 50$ piezas. La Figura 7(a) muestra una solución de valor $z = 27539$. El conjunto $\mathcal{L}$ está compuesto por $R_1 = (70, 41, 72, 81)$ y $R_2 = (72, 80, 100, 81)$. En el Paso 1 seleccionamos una pieza $i = 5$ de dimensiones (6x40) con $Q_i = 5$ y sólo 2 copias en la solución actual y consideramos un bloque B^* de una pieza. En el Paso 2 colocamos B^* sobre R_1, seleccionando la esquina superior izquierda del rectángulo para colocar la esquina superior izquierda del bloque. B^* cubre completamente R_1 y parte de R_2, que se transforma en $R_3 = (76, 80, 100, 81)$. B^* también cubre parcialmente un bloque de la solución (Figura 7(b)). Por tanto, en el Paso 3, eliminamos las piezas de la solución que se solapan con B^*. Esto produce dos nuevos rectángulos vacíos $R_4 = (76, 40, 78, 80)$ y $R_5 = (72, 40, 76, 41)$ (Figura 7(c)). En el Paso 4, el procedimiento constructivo comienza con la lista $\mathcal{L} = \{R_3, R_4, R_5\}$. Primero, R_3 y R_4 se fusionan, produciendo $R_6 = (76, 40, 78, 81)$ y $R_7 = (78, 80, 100, 81)$. Mientras que ninguna de las piezas restantes cabía en R_3 o en R_4, una pieza $i = 13$ de dimensiones (2x41) cabe en R_6. La nueva solución es mejor que la inicial y tiene un valor $z* = 27718$, que es el óptimo (Figura 7(d)).

5.2 Movimientos que se estudian

En cada iteración estudiamos todos los posibles movimientos de reducción e inserción que se pueden aplicar a la solución actual.

- Reducción:
  1. Tomar cada bloque de la solución, de uno en uno, en orden aleatorio.
  2. Considerar todas las posibilidades de reducción en las direcciones en las que sea adyacente a un rectángulo de pérdida.
- Inserción:
  1. Seleccionar una pieza para la que el número de copias en la solución, x_i, es menor que Q_i, de una en una, en orden aleatorio.



(a) Solución inicial (b) Insertar bloque

(c) Eliminar solapamiento (d) Fusionar y volver a cortar

Figura 7: *Inserción de bloques. Problema 1 de Fekete y Schepers (2004)*

2. Considerar todos los bloques posibles que se pueden formar con esa pieza.
3. Considerar todas las posiciones en las que se puede colocar el bloque sobre un rectángulo vacío.

5.3 Selección del movimiento

La función objetivo original consiste únicamente en maximizar el valor de las piezas cortadas $f(x) = \sum_i v_i x_i$. Sin embargo, si los movimientos se evalúan con esta función, pueden haber muchos movimientos con la misma evaluación. Para discriminar entre estos movimientos, utilizamos una función objetivo secundaria: $g(x) = k_1 S + k_2|\mathcal{L}| + k_3 C + k_4 F$.

- *S (Simetría)*: Intentamos no explorar soluciones simétricas y concentrarnos en aquellas soluciones en las que los rectángulos vacíos están concentrados preferentemente en la parte superior derecha del rectángulo.

$S = 1$ si no existe una solución simétrica con los rectángulos de pérdida más concentrados en la parte superior derecha. En otro caso, $S = 0$.

- $|\mathcal{L}|$ *(Número de rectángulos vacíos)).* Si es posible, preferimos soluciones con el menor número de rectángulos vacíos.
- *C (Rectángulos vacíos centrados y agrupados)*: Preferimos soluciones en las que los rectángulos vacíos están centrados y agrupados, ya que esto facilitará su fusión y poder cortar más piezas. Consideramos el menor rectángulo ER que contiene todos los rectángulos vacíos y

  $C = 1 - (0.75 * rd + 0.25 * ra)$

  donde rd es la distancia desde el centro de ER al centro del rectángulo inicial, dividida por la distancia del centro del rectángulo inicial a su esquina inferior izquierda, y ra es el área of ER dividida por el área del rectángulo inicial.
- *F (Factibilidad).* En problemas doblemente restringidos, la solución puede no ser factible. En ese caso $F = 1$. Si no, $F = 0$.

Estos criterios se suman en la función objetivo secundaria con pesos que reflejan su importancia relativa, de acuerdo con los resultados de un estudio computacional preliminar sobre un subconjunto de problemas. En la versión actual del algoritmo los pesos son:

Criterio	Coeficiente	Peso
Simetría	k_1	5000
Número de rectángulos vacíos	k_2	-950
Rectángulos vacíos centrados y agrupados	k_3	50
Factibilidad	k_4	-50000

5.4 Lista Tabú

La lista tabú contiene para cada solución un par de atributos: el valor de la función objetivo y el menor rectángulo ER que contiene todos sus rectángulos vacíos. Un movimiento es tabú si estos dos atributos de la nueva solución corresponden a un par de la lista tabú.

El tamaño de la lista tabú varía dinámicamente. Después de un cierto número de iteraciones sin mejorar la mejor solución conocida, la longitud de la lista se elige al azar en el intervalo $[0.25 * M, 0.75 * M]$, donde $M = \sum_i Q_i$.

El *criterio de aspiración* permite moverse a una solución con estatus tabú si mejora la mejor solución conocida.

5.5 Estrategias de intensificación y diversificación

Los movimientos que hemos definido implican un alto nivel de diversificación. Sin embargo, hemos añadido dos nuevas estrategias de diversificación:

- Memoria a largo plazo

  A lo largo del proceso de búsqueda, guardamos la frecuencia con la que cada tipo de pieza aparece en las soluciones.

  Esta información se usa para intensificación y diversificación. En una estrategia de diversificación, favorecemos movimientos que incluyen piezas que no aparecen frecuentemente en las soluciones. En una estrategia de intensificación, consideramos únicamente las piezas que aparecen en soluciones de alta calidad y favorecemos que esas piezas aparezcan en las nuevas soluciones.

  En la fase de diversificación, la función objetivo se modifica restándole un término que es la suma de las frecuencias de las piezas que aparecen en la solución

  $f(x) \rightarrow f(x) - \sum_i freq(p_i)$

  En la fase de intensificación, la función objetivo se modifica sumándole un término que es la suma de las frecuencias de las piezas que aparecen en un conjunto de soluciones de élite $\mathcal{E}$

  $f(x) \rightarrow f(x) + K \sum_{i \in \mathcal{E}} freq(p_i)$

- Reinicio

  De acuerdo con la función objetivo secundaria, tendemos a explorar soluciones que satisfagan el criterio de simetría. Después de un cierto número de iteraciones sin mejorar la mejor solución conocida, la solución actual se transforma aplicándole un movimiento de simetría vertical y otro horizontal respecto a los ejes vertical y horizontal que pasan por el centro del rectángulo. La nueva solución obtenida será muy diferente de las recientemente estudiadas y puede ser considerada como un punto de reinicio del proceso de búsqueda.

6 Estudio computacional

6.1 Problemas test

Hemos usado varios conjuntos de problemas test:

1. Un conjunto de 21 problemas de la literatura: 12 de Beasley (1985), 2 de Hadjiconstaninou y Christofides (1995), 1 de Wang (1983), 1 de Christofides

y Whitlock (1977), 5 de Fekete y Schepers (2004). Para todos ellos la solución óptima es conocida y aparece en Beasley (2004).

2. Un conjunto de 630 problemas grandes generados por Beasley (2004), siguiendo el esquema de Fekete y Schepers (2004). Todos tienen un rectángulo inicial $(100, 100)$. Para cada valor de m, número de tipos de piezas (m =40, 50, 100, 150, 250, 500, 1000), se han generado aleatoriamente 10 problemas con $P_i = 0$, $Q_i = Q^*, \forall i = 1, \ldots, m$ donde $Q^* = 1; 3; 4$. El valor asignado a cada pieza es igual a su área multiplicada por un entero aleatoriamente elegido entre $\{1, 2, 3\}$.

3. Los 21 problemas test mencionados en primer lugar fueron transformados por Beasley (2004) en problemas doblemente restringidos definiendo cotas inferiores P_i. Concretamente, para cada tipo de pieza $i = 1, \ldots, m$ que satisface:
   $\sum_{j=1, j\neq i}^{m} (l_j w_j) P_j + l_i w_i \leq (LW)/3$, la cota inferior P_i se fija a 1.
   Este conjunto de problemas nos permitirá probar nuestros algoritmos en el caso general de problemas doblemente restringidos.

4. Finalmente, hemos incluido los problemas test utilizados por Leung et al. (2003), entre los que se encuentran 3 problemas de Lai y Chan (199a), 5 de Jakobs (1996), y 2 de Leung et al. (2003). También hemos incluido 21 problemas más grandes de Hopper y Turton (2001). Son problemas en los que el valor de cada pieza corresponde a su área y el objetivo es minimizar los rectángulos vacíos asociados a la solución. Estos problemas han sido generados de forma que la solución óptima no contiene rectángulos vacíos.

Hemos incluido los problemas de Leung et al. (2003) y Hopper y Turton (2001) porque presentan características que los hacen complementarios de los problemas de los grupos anteriores. Mientras muchos de aquéllos pueden considerarse *problemas de selección*, ya que no todas las piezas posibles pueden ser cortadas y el problema es elegir el mejor subconjunto, éstos pueden considerarse *problemas rompecabezas*, pues todas las pieza caben en el rectángulo y el problema es elegir su posición.

6.2 Implementación de los algoritmos

Los algoritmos han sido codificados en lenguaje $C++$ y se han ejecutado en un ordenador $PentiumIII$ a $800\,Mhz$. En la implementación final del algoritmo GRASP utilizamos como estrategia de aleatorización la alternativa 1, basada en el Conjunto Restringido de Candidatos. Para la elección del δ utilizamos GRASP reactivo y como procedimiento de mejora el Método III. El algoritmo para cuando

se alcanza el límite de 10000 iteraciones o se iguala la cota inferior o la solución óptima si es conocida. Esta estrategia de parada cuando se consigue la solución óptima ha sido previamente utilizada por Beasley (2004) y la hemos adoptado para poder comparar con sus resultados.

En cuanto al algoritmo Tabu Search, el tamaño de la lista tabú cambia tras 100 iteraciones sin mejorar la mejor solución conocida. Si transcurren 400 iteraciones sin mejora, se realiza una fase de diversificación basada en memoria a largo plazo durante 100 iteraciones o hasta que se mejore la solución. Tras ella, se recupera la función objetivo original y se continúa el proceso de búsqueda. Después de otras 400 iteraciones sin mejora se realiza una fase de intensificación, con $K = 100$ durante 100 iteraciones o hasta que se mejore la solución. Tras ella recuperamos la función objetivo original, pero si la solución no ha mejorado, en lugar de continuar la búsqueda desde la solución actual se realiza un reinicio y se continúa a partir de la solución transformada. Como en el caso del algoritmo GRASP, el algoritmo Tabu Search para cuando se alcanza un límite de iteraciones, en este caso 1500, o cuando se llega a la solución óptima.

6.3 Resultados obtenidos

Los resultados obtenidos para los tres conjuntos de problemas restringidos aparecen en las tablas 1, 2 y 3. Los resultados sobre los problemas doblemente restringidos aparecerán más adelante. Las dos primeras tablas incluyen una comparación directa con los resultados obtenidos por Beasley (2004). Los tres algoritmos pueden compararse en términos de calidad. Sin embargo, los tiempos de ejecución no pueden compararse directamente. Beasley codificó su algoritmo en lenguaje FORTRAN y utilizó un ordenador Silicon Graphics O2 workstation (R10000 chip, 225MHz, 128 MB). Una comparación aproximada ($http://www.spec.org$) indica que su ordenador es el doble de rápido que el nuestro. En la tabla 1 vemos que el algoritmo Tabu Search resuelve óptimamente todos los problemas en tiempos de computación muy breves, aventajando a los otros algoritmos en términos de calidad y tiempo. Para los problemas grandes de la tabla 2 las soluciones óptimas no son conocidas y las comparaciones se hacen con las cotas superiores obtenidas resolviendo problemas mochila acotados (Sección 2). La tabla 2 muestra que el algoritmo Tabu Search obtiene nuevamente los mejores resultados sobre cada tipo de problemas, excepto para $m = 50$, $Q^* = 1$ donde GRASP es ligeramente mejor. Los tiempos de computación son mucho más cortos que los de Beasley, aunque son mayores que los requeridos por el GRASP. Ambos algoritmos están basados en ideas similares. El algoritmo Tabu Search, más complejo, obtiene mejores soluciones pero necesita tiempos de ejecución más largos. En la tabla 3 volvemos a comparar GRASP y Tabu Search. Tabu Search aventaja claramente a GRASP en cuanto a la calidad de las soluciones, con tiempos de ejecución similares.

La tabla 4 muestra los resultados de los algoritmos sobre el conjunto de pro-

blemas doblemente restringidos. La cota superior corresponde en este caso a la solución del problema restringido. Los problemas para los que los algoritmos no encuentran solución son claramente infactibles. Aparecen aquí para no modificar la tabla de resultados publicada por Beasley. El algoritmo Tabu Search obtiene los mejores resultados, pero sus tiempos de computación son más largos.

7 Bibliografía

[1] Alvarez-Valdes R, Parreño F, Tamarit JM. A GRASP algorithm for constrained two-dimensional non-guillotine cutting problems. Journal of the Operational Research Society 2005; 56; 414-425.

[2] Alvarez-Valdes R, Parreño F, Tamarit JM. A Tabu Search algorithm for two-dimensional non-guillotine cutting problems. European Journal of Operational Research 2005; en prensa.

[3] Amaral A, Letchford A. An improved upper bound for the two-dimensional non-guillotine cutting problem. Working paper available from the second author at Department of Management Science, Management School, Lancaster University, Lancaster LA1 4YW, England, 2003.

[4] Arenales M, Morabito R. An AND/OR-graph approach to the solution of two-dimensional non-guillotine cutting problems. European Journal of Operational Research 1995; 84; 599-617.

[5] Beasley JE. An exact two-dimensional non-guillotine cutting tree search procedure. Operations Research 1985; 33; 49-64.

[6] Beasley JE. A population heuristic for constrained two-dimensional non-guillotine cutting. European Journal of Operational Research 2004; 156; 601-627.

[7] Beltrán JC, Calderón JE, Cabrera RJ, Moreno JM. Procedimientos constructivos adaptativos (GRASP) para el problema del empaquetado bidimensional. Revista Iberoamericana de Inteligencia Artificial 2002; 15; 26-33.

[8] Caprara A, Monaci M. On the two-dimensional knapsack problem. Operations Research Letters 2004; 32; 5-14.

[9] Christofides N, Whitlock C. An algorithm for two-dimensional cutting problems. Operations Research 1977; 25; 30-44.

[10] Fekete SP, Schepers J. An exact algorithm for higher-dimensional orthogonal packing. Submitted to Operations Research 2004.

[11] Feo T, Resende MGC A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem, Operations Research Letters 1989; 8; 67-71.

[12] Glover F, Laguna M. Tabu Search. Kluwer Academic Publishers: Boston; 1997.

[13] Hadjiconstantinou E, Christofides N. An exact algorithm for general, orthogonal, two-dimensional knapsack problems. European Journal of Operational Research 1995; 83; 39-56.

[14] Healy P, Creavin M, Kuusik A. An optimal algorithm for placement rectangle. Operations Research Letters 1999; 24; 73-80.

[15] Hopper E, Turton BCH. An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. European Journal of Operational Research 2001; 128; 34-57.

[16] Jakobs S. On genetic algorithms for the packing of polygons. European Journal of Operational Research 1996; 88; 165-181.

[17] Lai KK, Chan JWM. Developing a simulated annealing algorithm for the cutting stock problem. Computers and Industrial Engineering 1997; 32; 115-127.

[18] Lai KK, Chan JWM. A evolutionary algorithm for the rectangular cutting stock problem. International Journal of Industrial Engineering 1997; 4; 130-139.

[19] Leung TW, Yung CH, Troutt MD. Applications of genetic search and simulated annealing to the two-dimensional non-guillotine cutting stock problem. Computers and Industrial Engineering 2001; 40; 201-214.

[20] Leung TW, Yung CH, Troutt MD. Application of a mixed simulated annealing-genetic algorithm heuristic for the two-dimensional orthogonal packing problem. European Journal of Operational Research 2003; 145; 530-542.

[21] Prais M, Ribeiro CC. Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. INFORMS Journal on Computing 2000; 12; 164-176.

[22] Resende MGC, Ribeiro CC. Greedy Randomized Adaptive Search Procedures. In Handbook of Metaheuristics, F.Glover and G.Kochenberger, Eds., Kluwer Academic Publishers 2003; 219-249.

[23] Scheithauer G, Terno J. Modeling of packing problems. Optimization 1993; 28; 63-84.

[24] Scheithauer G. LP-based bounds for the container and multi-container loading problem. International Transactions in Operations Research 1999; 6; 199-213.

[25] Tsai RD, Malstrom EM, Meeks HD. A two-dimensional palletizing procedure for warehouse loading operations. IIE Transactions 1988; 20; 418-425.

[26] Wäscher G, Haussner H, Schumann H. An improved typology of cutting and packing problems. European Journal of Operational Research 2006; en prensa.

[27] Wang PY. Two algorithms for constrained two-dimensional cutting stock problems. Operations Research 1983; 31; 573-586.

[28] Wu YL, Huang W, Lau SC, Wong CK, Young GH. An effective quasi-human based heuristic for solving rectangle packing problem. European Journal of Operational Research 2002; 141; 341-358.

Origen del problema	I	Tamaño (L,W)	m	M	Solución de Beasley	Solución GRASP	Solución TABU	Solución óptima	Tiempo de CPU (segundos) Beasley	GRASP	TABU
Beasley (1985)	1	(10, 10)	5	10	164	164	164	164	0,02	0,00	0,06
	2	(10, 10)	7	17	230	230	230	230	0,16	0,00	0,00
	3	(10, 10)	10	21	247	247	247	247	0,53	0,00	0,00
	4	(15, 10)	5	7	268	268	268	268	0,01	0,00	0,00
	5	(15, 10)	7	14	358	358	358	358	0,11	0,00	0,00
	6	(15, 10)	10	15	289	289	289	289	0,43	0,00	0,00
	7	(20, 20)	5	8	430	430	430	430	0,01	0,00	0,00
	8	(20, 20)	7	13	834	834	834	834	3,25	0,77	0,16
	9	(20, 20)	10	18	924	924	924	924	2,18	0,00	0,05
	10	(30, 30)	5	13	1452	1452	1452	1452	0,03	0,00	0,00
	11	(30, 30)	7	15	1688	1688	1688	1688	0,60	0,05	0,00
	12	(30, 30)	10	22	1801	1865	1865	1865	3,48	0,05	0,06
Hadjiconstantinou	3	(30, 30)	7	7	1178	1178	1178	1178	0,03	0,00	0,00
y Christofides (1995)	11	(30, 30)	15	15	1270	1270	1270	1270	0,04	0,00	0,00
Wang (1983)		(70, 40)	19	42	2721	2726	2726	2726	6,86	0,77	0,11
Christofides y Whitlock (1977)	3	(40, 70)	20	62	1720	1860	1860	1860	8,63	0,39	0,06
Fekete y Scheppers (2004)	1	(100, 100)	15	50	27486	27589	27718	27718	19,71	2,31	0,05
	2	(100, 100)	30	30	21976	21976	22502	22502	13,19	4,17	2,14
	3	(100, 100)	30	30	23743	23743	24019	24019	11,46	3,68	3,40
	4	(100, 100)	33	61	31269	32893	32893	32893	32,08	0,00	0,66
	5	(100, 100)	29	97	26332	27923	27923	27923	83,44	0,00	0,00
Porcentaje medio de desviación respecto al óptimo					1,21%	0,19%	0,00%		8,87	0,58	0,32
Número de soluciones óptimas (de 21)					13	18	21				

Tabla 1: *Resultados computationales – Problemas de la literatura*

Desviaciones medias respecto de la cota superior								
m	Q*	M	Solución de Beasley	Solución GRASP	Solución TABU	Tiempo de CPU (segundos) Beasley	GRASP	TABU
40	1	40	7,77	6,97	6,55	13,57	2,33	10,97
	3	120	3,54	2,22	1,95	47,43	6,62	14,20
	4	160	3,24	1,81	1,65	63,30	4,44	18,26
50	1	50	5,48	4,80	4,85	14,60	4,71	15,49
	3	150	2,35	1,50	1,27	59,27	7,05	22,50
	4	200	2,63	1,18	0,96	80,07	5,34	18,19
100	1	100	2,26	1,51	1,50	27,20	5,36	38,79
	3	300	1,27	0,47	0,31	119,47	9,41	32,11
	4	400	1,06	0,26	0,18	175,10	6,99	19,67
150	1	150	1,31	0,89	0,84	40,60	5,53	54,90
	3	450	0,60	0,14	0,07	190,53	11,71	31,76
	4	600	0,92	0,11	0,05	323,83	6,75	19,87
250	1	250	0,88	0,51	0,45	76,70	5,27	90,07
	3	750	0,57	0,04	0,01	439,47	13,89	13,70
	4	1000	0,39	0,03	0,00	693,67	6,65	4,50
500	1	500	0,26	0,05	0,03	203,10	3,24	86,17
	3	1500	0,18	0,00	0,00	1210,80	12,24	1,10
	4	2000	0,18	0,00	0,00	1790,83	1,15	0,84
1000	1	1000	0,09	0,00	0,00	667,23	1,01	7,80
	3	3000	0,07	0,00	0,00	3318,47	6,53	1,54
	4	4000	0,07	0,00	0,00	4840,57	0,29	1,19
Tipo I			1,64	1,04	0,95	558,11	5,13	19,61
Tipo 2			1,70	1,14	1,06	668,41	5,90	23,84
Tipo 3			1,66	1,03	0,94	830,02	7,28	32,56
Todos			1,67	1,07	0,98	685,51	5,91	25,34

Tabla 2: *Resultados computationales – Problemas aleatorios grandes*

Origen del problema	I	Tamaño (L,W)	m	M	Solución GRASP	Solución TABU	Solución óptima	Tiempo de CPU GRASP	Tiempo de CPU TABU
Lai y Chan (1997a)	1	(400,200)	9	10	80000	80000	80000	0,00	0,00
	2	(400,200)	7	15	79000	79000	79000	0,00	0,02
	3	(400,400)	5	20	154600	160000	160000	4,12	0,38
Jakobs (1996)	1	(70,80)	14	20	5447	5600	5600	10,16	1,89
	2	(70,80)	16	25	5455	5540	5600	15,44	16,88
	3	(120,45)	22	25	5328	5400	5400	12,57	0,42
	4	(90,45)	16	30	3978	4050	4050	10,28	1,97
	5	(65,45)	18	30	2871	2925	2925	14,94	1,53
Leung et al. (2003)	1	(150,110)	40	40	15856	16280	16500	90,52	52,36
	2	(160,120)	50	50	18628	19044	19200	132,26	63,95
Hopper y Turton (2001)	1-1	(20,20)	16	16	400	400	400	0,94	0,42
	1-2	(20,20)	17	17	386	400	400	9,28	4,23
	1-3	(20,20)	16	16	400	400	400	0,06	0,95
	2-1	(40,15)	25	25	590	600	600	19,44	0,44
	2-2	(40,15)	25	25	597	600	600	17,36	4,16
	2-3	(40,15)	25	25	600	600	600	0,71	0,00
	3-1	(60,30)	28	28	1765	1800	1800	26,80	4,91
	3-2	(60,30)	29	29	1755	1800	1800	37,35	10,11
	3-3	(60,30)	28	28	1774	1800	1800	30,92	5,52
	4-1	(60,60)	49	49	3528	3580	3600	102,05	45,27
	4-2	(60,60)	49	49	3524	3564	3600	110,79	68,59
	4-3	(60,60)	49	49	3544	3580	3600	94,41	51,11
	5-1	(60,90)	73	73	5308	5342	5400	212,07	135,97
	5-2	(60,90)	73	73	5313	5361	5400	231,56	96,80
	5-3	(60,90)	73	73	5312	5375	5400	231,24	82,06
	6-1	(80,120)	97	97	9470	9548	9600	480,44	240,39
	6-2	(80,120)	97	97	9453	9448	9600	465,49	399,86
	6-3	(80,120)	97	97	9450	9565	9600	478,02	206,78
	7-1	(160,240)	196	196	37661	38026	38400	3760,14	3054,38
	7-2	(160,240)	197	197	37939	38145	38400	2841,96	1990,70
	7-3	(160,240)	196	196	37745	37867	38400	3700,99	5615,75
Porcentaje medio de desviación del óptimo					1,68%	0,42%		423,95	392,19
Número de soluciones óptimas (de 31)					5	16			

Tabla 3: *Resultados computationales – Problemas sin pérdida*

Origen del problema	I	Tamaño (L,W)	m	M	Solución de Beasley	Solución GRASP	Solución TABU	Cota superior	Tiempo de CPU (segundos) Beasley	GRASP	TABU
Beasley (1985)	1	(10, 10)	5	10	164	164	164	164	0,02	0,00	0,00
	2	(10, 10)	7	17	225	225	225	230	5,53	0,71	1,70
	3	(10, 10)	10	21	220	220	220	247	7,85	1,21	2,26
	4	(15, 10)	5	7	268	268	268	268	0,01	0,00	0,00
	5	(15, 10)	7	14	301	301	301	358	5,05	0,72	1,48
	6	(15, 10)	10	15	265	252	265	289	6,81	1,81	1,59
	7	(20, 20)	5	8	430	430	430	430	0,01	0,00	0,00
	8	(20, 20)	7	13	819	819	819	834	6,54	1,32	1,76
	9	(20, 20)	10	18	924	924	924	924	5,64	0,00	0,00
	10	(30, 30)	5	13	n/f	n/f	n/f	n/f	2,38	0,22	0,94
	11	(30, 30)	7	15	1505	1518	1518	1688	2,96	1,59	2,52
	12	(30, 30)	10	22	1666	1648	1672	1865	3,78	1,65	3,73
Hadjiconstantinou	3	(30, 30)	7	7	1178	1178	1178	1178	0,25	0,00	0,00
y Christofides (1995)	11	(30, 30)	15	15	1216	1216	1216	1270	2,60	2,08	3,18
Wang (1983)		(70, 40)	19	42	2499	2700	2716	2726	6,36	1,48	6,16
Christofides y Whitlock (1977)	3	(40, 70)	20	62	1600	1720	1720	1860	6,81	0,88	5,27
Fekete y Scheppers (2004)	1	(100, 100)	15	50	25373	24869	25384	27718	11,86	3,73	25,27
	2	(100, 100)	30	30	17789	19083	19657	22502	5,80	3,02	18,35
	3	(100, 100)	30	30	n/f	n/f	n/f	n/f	4,03	0,66	12,41
	4	(100, 100)	33	61	27556	27898	28974	32893	20,42	2,80	37,46
	5	(100, 100)	29	97	21997	22011	22011	27923	18,41	3,30	61,90
Porcentaje medio de desviación de la cota superior					8,11%	7,36%	6,62%		5.86	1.29	8.86

n/f: Problema no factible

Tabla 4: *Resultados computacionales – Problemas doblemente restringidos*

Precedimientos heurísticos para la secuenciación de proyectos con recursos parcialmente renovables

R. Alvarez-Valdes[a], **E. Crespo**[b], **J.M. Tamarit, F. Villa**[a]

[a] Departament d'Esdadística i Investigació Operativa
Universitat de València

[b] Departament de Matemàtiques per a l'Economia i l'Empresa.
Universitat de València

1 Introducción

La secuenciación de proyectos consiste en determinar el inicio y final de un conjunto de actividades en un proyecto. Dichas actividades están ligadas entre sí por relaciones de precedencia y requieren uno o más recursos. La secuenciación de proyectos ha sido objeto de una gran aten-ción en la investigación desde que los primeros métodos, CPM y PERT, fueron desarrollados en los años 50. Estos procedimientos fueron capaces de resolver grandes problemas y fueron considerados una herramienta muy útil en el proceso de planificación. Sin embargo, ambos presuponen que los recursos son ilimitados con lo que su aplicación queda muy limitada en los problemas reales. Por esto muchos investigadores comenzaron a estudiar el caso de los recursos limitados (RCPSP) del cual se han desarrollado hasta el momento muchos algoritmos exactos y aproximados. El libro de Demeulemeester y Herroelen [4] muestra una excelente descripción de la actual situación de la investigación. El problema clásico del RCPSP incluye dos tipos de recursos; los renovables, cuya disponibilidad se renueva en cada período del intervalo de planificación, y los no renovables, cuya disponibilidad se va reduciendo a lo largo

del proyecto a medida que se consumen. Sin embargo estos dos tipos de recursos no son suficientes para representar muchas situaciones reales por lo que han sido propuestos algunos otros tipos de recursos como los comprometidos (*allocatable*) [8, 13] o los acumulativos[9, 10].

En nuestro trabajo consideramos los recursos parcialmente renovables, introducidos por Böttcher et al. [3] en 1999. La disponibilidad de este tipo de recursos está asociada a un subconjunto de periodos del horizonte de planificación y las actividades sólo consumen el recurso si son procesadas dentro de esos periodos. Este tipo de recursos pueden ser un instrumento poderoso para resolver problemas de secuenciación de actividades. Además, desde el punto de vista teórico, engloban como casos particulares tanto a los renovables como a los no renovables. Por otro lado, los recursos parcialmente renovables permiten modelizar complicadas reglas laborales y restricciones en problemas de horarios como casos concretos de problemas de secuenciación. Como ejemplo consideremos un proyecto que involucra recursos humanos. Podemos encontrarnos con condiciones laborales tales como trabajar a lo sumo dos días del fin de semana cada tres semanas consecutivas. Esta restricción no puede ser modelizada como un recurso renovable porque eso exigiría considerar cada periodo por separado. Tampoco puede serlo como uno no renovable porque tendríamos que considerar todo el horizonte de planificación. Sin embargo podemos hacerlo como un recurso parcialmente renovable con un conjunto de periodos 6, 7, 13, 14, 20, 21 que incluyen los días de los tres primeros fines de semana y con una disponibilidad de 2 unidades. Cada tarea consume 1 unidad de dicho recurso durante cada día del fin de semana en el que es procesada. Supongamos tres actividades A, B y C secuenciadas en la escala temporal. De ellas, la actividad A está en proceso en los periodos 5, 6 y 7 y, por tanto, consume 2 unidades del recurso. La actividad B se realiza en los periodos del 9 al 12, ambos inclusive, y no consume nada. Por fin, la actividad C comienza en el 16 y acaba en el 20 y consume únicamente 1 unidad en el periodo 20. Si estas tres tareas fueran realizadas por el mismo trabajador la solución sería imposible porque excede a la disponibilidad de recursos.

Böttcher et al.[3] propusieron una formulación entera y desarrollaron algoritmos exactos y heurísticos. Schirmer [12] estudió ampliamente este nuevo tipo de recursos en su libro sobre problemas de secuenciación de actividades. En él presentó muchos ejemplos de condiciones especiales que podían ser adecuadamente modelizadas usando los recursos parcialmente renovables y también presentó varias familias de algoritmos aproximados para resolver el problema de secuenciación de actividades con recursos parcialmente renovables (RCPSP/π). En este trabajo describimos algunas técnicas de preproceso y desarrollamos diferentes algoritmos heurísticos para el RCPSP/π. El preproceso reduce las dimensiones del problema tanto en los recursos como en los tiempos posibles de finalización, a partir de aquí TPF, de las actividades y, por tanto, mejora la eficiencia de los algoritmos.

En el apartado 2 definimos los elementos del problema y exponemos una for-

mulación entera del mismo. El apartado 3 describe los aspectos principales del preproceso. En el 4 exponemos el algoritmo constructivo que utilizaremos en las metaheurísticas de los apartados posteriores. Los apartados 5 y 6 están dedicados a exponer los procedimientos GRASP y de Búsqueda Dispersa respectivamente, estableciendo comparaciones entre los resultados obtenidos por ambos. Por fin, el apartado 7 expone conclusiones y futuras líneas de investigación.

2 Formulación del problema

El RCPSP/π puede ser definido de la manera siguiente: Sea J el conjunto de $n = |J|$ actividades, numeradas de 1 a n, donde las actividades ficticias 1 y n representan el inicio y final del proyecto. Sea P_j el conjunto de actividades que son predecesoras inmediatas de la actividad j y P'_j el conjunto de todas las predecesoras de j. Cada actividad j tiene una duración d_j y no puede ser interrumpida. Sea R el conjunto de recursos parcialmente renovables. Cada recurso $r \in R$ tiene una disponibilidad total K_r y un conjunto de periodos asociado Π_r. Una actividad j que necesite del recurso r consumirá k_{jr} unidades del mismo en cada periodo $t \in \Pi_r$ en que esté en proceso. Finalmente, sea T el último periodo del horizonte de planificación. Para cada actividad j , mediante el análisis del camino crítico, obtenemos el primero y el último de los TPF de la actividad que denotamos EFT_j y LFT_j. Representamos por $E_j = \{EFT_j, ..., LFT_j\}$, su conjunto de TPF, y por $Q_{jt} = \{t, ..., t + d_j - 1\}$.
El RCPSP/π consiste en secuenciar las actividades de manera que se satisfagan las relaciones de precedencia y las restricciones de los recursos y se minimize el tiempo total de duración del proyecto o *makespan*.

Si definimos las variables:

$$x_{jt} = \begin{cases} 1 & \text{si la actividad } j \text{ acaba en el tiempo } t \\ 0 & \text{en otro caso.} \end{cases}$$

se puede formular como:

$$Min \quad \sum_{t \in E_n} t x_{nt} \tag{1}$$

$$s.t. \quad \sum_{t \in E_j} x_{jt} = 1 \qquad j \in J \tag{2}$$

$$\sum_{t \in E_i} t x_{it} \leq \sum_{t \in E_j} (t - d_j) x_{jt} \qquad j \in J,\ i \in P_j \tag{3}$$

$$\sum_{j \in J} k_{jr} \sum_{t \in \Pi_r} \sum_{q \in Q_{jt} \bigcap E_j} x_{jq} \leq K_r \qquad r \in R \tag{4}$$

$$x_{jt} \in \{0,1\} \qquad j \in J,\ t \in E_j \qquad (5)$$

La función objetivo (1) minimiza el tiempo final de la última actividad y, por tanto, la duración del proyecto. Las restricciones (2) aseguran que cada actividad acaba una sola vez. Las restricciones (3) son las de precedencia y (4) las de recursos. Nótese que a diferencia del problema con recursos renovables donde hay una restricción por cada recurso y periodo, en este problema sólo hay una restricción global para cada recurso $r \in R$. Otra característica especial de este problema es que todas las actividades deben acabar dentro del intervalo cerrado E_j porque el conjunto Π_r está definido dentro del horizonte de planificación $(0, T)$. Por tanto, no está garantizada la existencia de soluciones posibles. De hecho, Schirmer[12] ha probado que el problema de factibilidad del RCPSP/π es NP-completo en sentido estricto.

La formulación anterior se conoce como formulación normalizada de Böttcher et al. [3] y Schimer[12]. En sus trabajos también han considerado formulaciones alternativas pero finalmente adoptaron la normalizada por su simplicidad.

3 El preproceso

El preproceso tiene dos objetivos. En primer lugar, ayudar a decidir si una instancia dada tiene soluciones posibles. En este caso, el segundo objetivo es reducir la cantidad de recursos y de TPF de las actividades. Si se consiguen estos dos objetivos, los procedimientos de solución no perderán tiempo tratando de resolver problemas imposibles y concentrarán sus esfuerzos en los elementos significativos del problema. El preproceso que hemos desarrollado incluye varios procedimientos:

1. *Identificar los problemas triviales.* Se secuencia cada actividad j en su $EFTj$ Si esta solución cumple las restricciones de recursos, es la solución óptima.

2. *Reducir el horizonte de planificación.* Construimos una primera solución aplicando un algoritmo iterativo GRASP. Si el *makespan* de esta solución, t_S, coincide con el que proporcionaría el CPM, la solución es óptima y el preproceso finaliza. En caso contrario, hacemos $T = t_S$. Con este nuevo T se recalculan los $LFTj$ reducidos y con ellos los intervalos Ej.

3. *Eliminar recursos ociosos y no restrictivos.* En primer lugar, eliminamos recursos ociosos, es decir, aquellos recursos r que no son utilizados por ninguna actividad. Esto sucede si $\forall j \in J$ con $k_{rj} \neq 0 \rightarrow \Pi_r \bigcap Ej = \emptyset$. A continuación, calculamos para cada uno de los recursos no eliminados una cota superior de la cantidad máxima que de él se demanda resolviendo el siguiente problema lineal:

$$Max \qquad DMD_r = \sum_{j=2}^{J-1} k_{jr} \sum_{t \in \Pi_r} \sum_{q \in Q_{jt} \bigcap E_j} x_{jq} \tag{6}$$

$$s.t. \qquad \sum_{t=EFT_j}^{LFT_j} x_{jt} = 1 \qquad (1 \leq j \leq J) \tag{7}$$

$$\sum_{m=t}^{T} x_{im} + \sum_{s=1}^{t+d_j-1} x_{js} \leq 1 \qquad (j \leq J; i \in P_j; t \leq T) \tag{8}$$

$$x_{jt} \leq 0 \qquad (1 \leq j \leq J; EFT_j \leq t \leq LFT_j) \tag{9}$$

Se trata de maximizar la demanda de cada recurso (6), sujeto a que cada actividad debe acabar en un y sólo en un instante (7), se respeten las relaciones de precedencia (8), y todas las variables son no negativas. Si este máximo no supera la disponibilidad del recurso, no será restrictivo y puede eliminarse.

4. *Eliminación de variables.* Calculamos, para cada uno de los TPF de cada actividad j y cada recurso, el consumo que realizaría esa actividad si acabase en ese tiempo junto con el consumo que como mínimo realizarían el resto de actividades teniendo en cuenta que j acaba en ese TPF. Si esta estimación del consumo global excede la disponibilidad del recurso, el TPF no será posible y es eliminado. Al eliminar un TPF de una actividad se comprueba la coherencia de los TPF de una actividad con los TPF de sus antecesoras y sus sucesoras. Si alguno de éstos es imposible, se suprime. Si como consecuencia de estos procesos alguna actividad se queda sin TPF el problema es imposible.

Todos estos pasos pueden encontrarse expuestos detalladamente en [1, 2].

4 El procedimiento GRASP

El GRASP (Greedy Randomized Adaptive Search Procedure) es un proceso iterativo que combina una fase constructiva y una fase de mejora hasta que se cumple un criterio de parada. La fase constructiva construye una solución paso a paso, añadiendo elementos a una solución parcial. El elemento a añadir es seleccionado de acuerdo con una función *greedy* que se adapta dinámicamente

según se va construyendo la solución. Sin embargo la selección que se hace no es determinista sino sujeta a una cierta aleatorización. Por tanto, cuando repetimos el proceso obtenemos diferentes soluciones. Cuando hemos obtenido una solución posible exploramos su entorno en una fase de búsqueda local hasta que obtenemos un óptimo local. Las primeras dos subsecciones contienen el algoritmo constructivo y la fase de mejora. Las dos últimas describen procedimiento de GRASP agresivo y un Reencadenamiento de Trayectorias (*Path Relinking*) que opera sobre las mejores soluciones obtenidas por el GRASP. Se puede encontrar una visión general sobre el GRASP en Resende y Ribeiro[11] y un compendio extensivo sobre la literatura acerca del GRASP en Festa and Resende[5].

4.1 El algoritmo constructivo

Hemos adaptado el Esquema de Secuenciación en Serie (SSS) propuesto por Schirmer[12], que a su vez es una adaptación del Esquema de Secuenciación en Serie normalmente usado para el clásico RCPSP. Denotamos por FT_j el tiempo de finalización asignado a la actividad j. En cada etapa del procedimiento iterativo se secuencia una actividad eligiendo entre el vigente conjunto de decisiones, que son pares (j,t) formados por una actividad j y un tiempo posible de finalización $t \in TPF_j$. La selección se basa en una regla de prioridad aleatorizada.

Paso 0. Inicialización

$s = 1$ (pone en marcha el contador)

$FT_1 = 0$ (secuenciamos la actividad ficticia 1)

$S_1 = \{1\}$ (secuenciación parcial en la etapa 1)

EL_1= conjunto de actividades elegibles (actividades que tienen a 1 por único predecesor)

Paso 1. Construcción del conjunto de decisiones

$D_s = \{(j,t) \,|\, j \in EL_s \,,\, t \in TPF_j\}$

Paso 2. Elección de la decisión

Seleccionamos una decisión (j^*, t^*) in D_s, de acuerdo con una regla aleatorizada de prioridad

Paso 3. Test de Factibilidad

Si (j^*, t^*) es factible por recursos, ir al Paso 4

En caso contrario

$D_s = D_s \setminus \{(j^*, t^*)\}$

Si $D_s = \emptyset$, llamar al Mecanismo de Reparación

Si encontramos una decisión posible (j^*, t^*) para secuenciar $j^* \in EL_s$, ir al Paso 4

En otro caso, PARAR. El algoritmo no encuentra una solución posible

En caso contrario, ir al Paso 2

Paso 4. Actualización

$s = s + 1$

$FT_{j^*} = t^*$

$S_s = S_{s-1} \cup \{j^*\}$

$EL_s = (EL_{s-1} \setminus \{j^*\}) \cup \{j \in J \mid P_j \subseteq S_s\}$

$\forall l \in J \mid j \in P_l : TPF_l = TPF_l \setminus \{\tau \mid t^* + d_l > \tau\}$

Si $s = n$, PARAR. La secuenciación se ha completado.

En caso contrario, ir al Paso 2.

En el Paso 1, la construcción de D_s podría haber incluido el test de factibilidad del Paso 3, como en el esquema original de Schirmer [12]. Sin embargo, hemos preferido no comprobar la disponibilidad de recursos de cada decisión sino solamente la de la decisión que escogemos. En los problemas con un gran número de TPF para las actividades esta estrategia es más eficiente.

Para seleccionar la regla de prioridad hemos probado con 32 reglas de prioridad usadas por Schirmer[12]. Las 8 primeras están basadas en la estructura de la red, incluyendo las reglas clásicas como EFT, LFT, SPT o MINSLK. La otras 24 reglas se basan en la utilización de los recursos. 12 de ellas usan todos los recursos y las otras 12 solamente los escasos. Una experiencia computacional preliminar nos condujo a elegir la regla LFT como la más adecuada en términos de velocidad y calidad de la solución. Estos resultados previos también mostraban que incluso con las reglas más eficientes el algoritmo constructivo, si seleccionábamos en cada etapa la decisión con mayor prioridad, fallaba para obtener una solución posible para muchas de las instancias de 10 actividades generadas por Böttcher et al.[3]. Por tanto, la aleatorización incluida en el algoritmo no persigue sólo conseguir soluciones diversas, sino también asegurar la obtención de soluciones posibles para la mayoría de los problemas.

Introducimos un procedimiento de aleatorización para seleccionar la decisión en el Paso 2. Sea s_{jt} la puntuación de la decisión (j, t) en la regla de prioridad y $s_{max} = max\{s_{jt} | (j, t) \in D_s\}$, y sea δ un parámetro a determinar $(0 < \delta < 1)$. Hemos considerado tres alternativas:

1. *Elección equiprobable en una Lista Restringida de Candidatos, S*

   Elegimos una decisión (j^*, t^*) al azar en el conjunto $S = \{(j, t) \mid s_{jt} \geq \delta s_{max}\}$

2. *Elección sesgada en una Lista Restringida de Candidatos, S*

   Las decisiones que incumben a la misma actividad j reciben un peso que es inversamente proporcional al orden de sus tiempos de finalización. Por ejemplo, si en S tenemos las decisiones $(2,4),(2,5),(2,7),(2,8)$ que incumben a la actividad 2 y las ordenamos por sus tiempos de finalización crecientes, entonces la decisión $(2,4)$ tendrá un peso igual a 1, la decisión $(2,5)$ un peso de $1/2$, la $(2,7)$ un peso de $1/3$ y la $(2,8)$ de $1/4$. El mismo método se aplica a las decisiones correspondientes a las otras actividades. Por tanto, las decisiones de S correspondientes a los tiempos de finalización más bajos de las actividades involucradas serán equiprobables y el proceso de selección aleatoria les favorecerá.

3. *Elección sesgada en el conjunto de decisiones D*

   La decisión (j,t) se elige entre todas las de D con una probabilidad que viene dada por un valor de Regret propuesto por Schirmer [12].

Los resultados de las pruebas computacionales, que pueden encontrarse en [1], nos hicieron decidirnos por el segundo procedimiento para implementar el algoritmo definitivo.

Constatamos que la estrategia aleatoria mejora significativamente la habilidad del algoritmo constructivo para encontrar soluciones posibles para instancias fuertemente restringidas. Sin embargo, una limitada experiencia computacional nos ha mostrado que el algoritmo construtivo podría ser incapaz de obtener soluciones posibles para todas las instancias de 10 actividades generadas por Böttcher et al.[3]. Por tanto, decidimos incluir un mecanismo de reparación para secuencias parciales infactibles. Si en el Paso 3 todas las decisiones de D_n fallan en la prueba de factibilidad y D_n resulta vacío, en lugar de parar el proceso e iniciar una nueva iteración tratamos de reasignar algunas de las actividades ya secuenciadas a otros TPF con la idea de liberar algunos recursos que puedan ser utilizados para que alguna de las actividades todavía no secuenciadas sea procesada. Si este procedimiento tiene éxito, el proceso constructivo continua. En caso contrario, se para.

4.2 La fase de mejora

Dada una solución posible obtenida en la fase constructiva, la fase de mejora consta de 2 pasos. Primero, identificar las actividades cuyos tiempos de finalización deban ser reducidos para conseguir una nueva solución con el menor tiempo tiempo total, que etiquetamos como *críticas*. Después, mover las actividades críticas de manera que la secuencia resultante sea posible de acuerdo con las relaciones de precedencia y recursos. Hemos diseñado dos tipos de movimientos: un movimiento simple, que implica una sola actividad crítica, y un

movimiento doble en el cual, además de la actividad crítica, también son movidas otras actividades.

1. *Construcción de M, el conjunto de las actividades críticas*

   Paso 0. Inicialización

   $M = \{n\}$ (la última actividad del proyecto, n, siempre es crítica)

   Paso 1. Añadir actividades a M

   Mientras($\exists j \in M$ que no haya sido considerada para aumentar M)

   {

   Tomar la actividad más larga todavía no considerada $j \in M$

   $\forall i \in P_j$: Si $FT_i + d_j = FT_j$ (no hay holgura entre i y j)

   $M = M \cup \{i\}$

   }

   En el Paso 1, la condición para incluir una actividad en M es, simplemente, si j ha de ser movida hacia la izquierda, reduciendo su tiempo de finalización, un predecesor i procesado inmediatamente antes de j también ha de ser movido hacia la izquierda para dejar espacio para poder mover j. Esta condición, si consideramos que los filtros del preproceso pueden haber eliminado algunos TPF de las actividades, puede ser refinada. Si $t'_j = max\{t \in TPF_j \,|\, t'_j < FT_j\}$, la condición del Paso 1 puede escribirse como: Si $FT_i + d_j > t'_j$, i es crítica.

2. *Movimiento simple*

   Intentamos mover, en orden topológico, cada actividad $j \in M$ hacia la izquierda a un nuevo tiempo de finalización en el que satisfaga las relaciones de precedencia y recursos. Si alguna actividad no se puede mover, el procedimiento se detiene. Si hay varios nuevos TPF para una actividad, elegimos aquel cuya suma de todos los consumos de recursos sea mínima.

3. *Movimiento doble*

   Consideramos las actividades de M, en orden topológico, para valorar sus posibilidades de ser movidas hacia la izquierda. Para cada actividad $j \in M$ estudiamos todos los TPF que sean anteriores al vigente. Si encontramos un t factible por recursos, movemos j para que acabe en t y no es necesario mover ninguna otra actividad. En caso contrario, consideramos la posibilidad de mover las otras actividades de J. Movemos una actividad i, hacia la izquierda o la derecha, a un nuevo tiempo de finalización provisional si satisface las relaciones de precedencia y compensa la violación de recursos provocada por el movimiento de j o, al menos, reduce el déficit.

Por tanto, a lo largo de la búsqueda en J, construimos una lista provisional de cambios LC hasta que la solución es reparada o J está agotada. Si la solución es reparada con la lista LC, efectuamos estos cambios y consideramos un nuevo $j \in M$. En caso contrario, el procedimiento se para sin mejorar la solución.

El movimiento doble se puede reforzar utilizándolo repetidamente mientras se produzcan reducciones del déficit. El procedimiento es más complejo pero, a veces, ofrece movimientos posibles para las actividades críticas.

Los tres procedimientos de la fase de mejora se realizan iterativamente:

S= solución vigente

$improve = false$

do{

Construir el conjunto M de actividades críticas

$improve$=SimpleMove(S, M)

if $improve = false$

$improve$=DoubleMove(S, M)

} while ($improve = true$)

4.3 Un procedimiento agresivo

La versión estandar de nuestro algoritmo heurístico comienza aplicando el preproceso de la Sección 3. El problema reducido va entonces al algoritmo GRASP descrito antes, combinando una fase constructiva y una de mejora en cada iteración, hasta que el procedimiento de parada, en este caso un número fijo de iteraciones, lo interrumpe.

Una versión reforzada del algoritmo heurístico combina el preproceso y el GRASP de una forma más *agresiva*. Después de un número determinado de iteraciones (criterio de parada), comprobamos si la mejor solución conocida ha mejorado. En este caso, fijamos el horizonte T en el nuevo *makespan* mejorado y volvemos a pasar los filtros de reducción de variables. El algoritmo GRASP se aplica entonces al problema reducido. Ahora, obtener soluciones posibles es más difícil, pero si las conseguimos serán de alta calidad.

4.4 Reencadenamiento de Trayectorias

Si durante los procedimientos descritos antes guardamos un conjunto de buenas soluciones, normalmente llamadas *soluciones de élite*, podemos llevar a cabo un procedimiento de Reencadenamiento de Trayectorias. Comenzando en una de estas soluciones de élite, llamada *solución inicial*, construimos un camino

hacia otra solución de élite, llamada *solución guía*. En el camino vamos imponiendo progresivamente, a las soluciones intermedias, los atributos de la solución guía de manera que evolucionen desde la solución inicial hasta que alcancen la solución guía. A lo largo de este camino confiamos en encontrar soluciones que sean mejores que las dos soluciones de élite que estamos utilizando.

En nuestro caso guardamos las 10 mejores soluciones obtenidas en el GRASP y combinamos todas las parejas posibles tomando una de ella como solución inicial y la otra como guía y, después, al contrario. Dada una solución inicial se le va imponiendo sucesivamente a cada actividad el tiempo de finalización correspondiente de la solución guía. Así vamos obteniendo una familia de soluciones que se transforman de la inicial a la guía. En muchos casos estas soluciones intermedias no son posibles. Entonces aplicamos un mecanismo de reparación similar al descrito en la Sección 4.1. Vamos de la actividad 1 a la n comprobando, para cada actividad j, si la solución parcial de 1 a j es posible. Si no lo es, tratamos de repararla. Si lo conseguimos pasamos a estudiar la actividad $j + 1$. En otro caso, descartamos la solución y procedemos con la solución intermedia siguiente. Si obtenemos una solución intermedia completa que sea posible le aplicamos la fase de mejora descrita en el GRASP.

5 El procedimiento de Scatter Search

Un algoritmo de Scatter Search es un procedimiento aproximado donde se construye una población de soluciones posibles y, posteriormente, se combinan sistemáticamente los elementos de unos subconjuntos determinados con la finalidad de producir nuevas soluciones posibles que esperamos que mejoren la mejor solución conocida (para una descripción más exhaustiva del algoritmo se puede ver el libro de Laguna and Marti[7]). El esquema básico del algoritmo se compone de 5 pasos:

1. Generación y mejora de soluciones
2. Construcción del Conjunto de Referencia
3. Elección de Subconjuntos
4. Procedimiento de combinación
5. Actualización del Conjunto de Referencia

Este algoritmo básico acaba cuando el Conjunto de Referencia no puede ser actualizado y no hay nuevas soluciones disponibles para el procedimiento de combinación. Sin embargo, el esquema puede ser intensificado añadiendo un nuevo paso donde el Conjunto de Referencia es regenerado y, por tanto, son posibles nuevas combinaciones. Las subsecciones siguientes describen con detalle cada paso del algoritmo.

5.1 Generación y mejora de soluciones

En nuestro algoritmo la población inicial se genera mediante una versión simplificada del algorimo GRASP expuesto anteriormente.

5.2 Construcción del Conjunto de Referencia

A partir de la población inicial seleccionamos un conjunto de b soluciones para formar el Conjunto de Referencia, S, que será el conjunto cuyas soluciones combinaremos para obtener nuevas soluciones. La estrategia habitual es seleccionar b_1 elementos con el criterio de calidad: las b_1 soluciones con los *makespans* más cortos y en el caso de empates eligiendo aleatoriamente. Las restantes $b_2 = b - b_1$ soluciones se seleccionan con el criterio de diversidad: las soluciones se seleccionan una por una, buscando en cada momento la solución más lejana a las soluciones que integran en ese momento el Conjunto de Referencia. Esto es, seleccionamos la solución s^* para la cual el $Min_{s \in S}\{dist(s, s^*)\}$ es máximo. Definimos la distancia entre dos soluciones s_1 y s_2 como

$$dist(s_1, s_2) = \sum_{i=1}^{n} |s_1^i - s_2^i|$$

donde s_j^i es el tiempo de finalización de la i-ésima actividad de la solución s_j.

5.3 Elección de Subconjuntos

Hemos desarrollado y comprobado diferentes procedimientos de combinación. La mayoría de ellos combina 2 soluciones, mientras que uno combina 3. La primera vez que ejecutamos la combinación se combinan todos los pares (o trios). En las ejecuciones siguientes, cuando el Conjunto de Referencia ha sido actualizado y se compone de soluciones viejas y nuevas, sólo estudiamos las combinaciones que contienen, al menos, una solución nueva.

5.4 Procedimiento de combinación de soluciones

Hemos desarrollado ocho procedimientos diferentes de combinación. Cada solución s_j está representada por el vector de tiempos de finalización de cada una de las n actividades del proyecto: $s_j = (s_j^1, s_j^2, ..., s_j^n)$. Cuando combinamos 2 soluciones s_1 y s_2 (o 3 soluciones s_1, s_2 y s_3), ordenamos las soluciones según los *makespans* no decrecientes. Por tanto, s_1 será una solución con un *makespan* menor o igual que s_2 (y el *makespan* de s_2 será menor o igual que el de s_3).

En este trabajo sólo exponemos las combinaciones 1 y 8 que han resultado las más eficaces y sobre las que se implementaron las mejoras posteriores como fue la regeneración del conjunto de referencia. Además son las que vienen reflejadas en los resultados computacionales. La descripción de las otras combinaciones puede encontrarse en [2].

Combinación 1

Los tiempos de finalización de cada actividad en la nueva solución, s_c, se obtienen como una media ponderada de los tiempos de finalización de las dos soluciones, con pesos relacionados con sus *makespans* según la expresión:

$$s_c^i = \lfloor \frac{k_1 s_1^i + k_2 s_2^i}{k_1 + k_2} \rfloor \quad \text{donde } k_1 = (1/s_1^n)^2 \text{ y } k_2 = (1/s_2^n)^2$$

Combinación 8

Se combinan tres soluciones s_1, s_2 y s_3 mediante un procedimiento de votación. Para decidir el valor de s_c^i, las tres soluciones votan cada una por su propio tiempo de finalización s_1^i, s_2^i, s_3^i. Escogemos como s_c^i el valor con mayor número de votos. Si los tres valores son diferentes tendremos un empate. En este caso, si el *makespan* de s_1 es estrictamente menor que los otros actúa como voto de calidad e impone su tiempo de finalización a los otros. Si no lo fuera, es decir si dos o tres soluciones tienen el mismo *makespan* mínimo el tiempo de finalización se escoje al azar entre aquellos que corresponden a las soluciones empatadas.

La mayoría de las soluciones obtenidas por procedimientos de combinación no satisfacen todas las restricciones de precedencia y recursos. Las soluciones imposibles son sometidas a un proceso de reparación que trata de conseguir soluciones posibles tan próximas como sea posible a las soluciones creadas por la combinación. Este proceso se compone de dos fases. En la primera consideramos los tiempos de finalización s_c^i en orden topológico para comprobar si la solución parcial $(s_c^1, s_c^2, .., s_c^i)$ satisface todas las restricciones. En este caso, estudiamos el tiempo siguiente s_c^{i+1}. En caso contrario, descartamos s_c^i como tiempo de finalización de la actividad i y buscamos un nuevo tiempo entre aquellos que son posibles para i. La búsqueda va de los tiempos más próximos a s_c^i a los más lejanos. Cuando encontramos un tiempo t^i que puede ser incluido en una solución parcial $(s_c^1, s_c^2, .., t^i)$ paramos la búsqueda y consideramos el tiempo siguiente s_c^{i+1}. Si no encontramos ningún tiempo posible para la actividad i el proceso va a una segunda fase que consiste en un procedimiento de reparación similar al del algoritmo constructivo que trata de cambiar el tiempo de finalización de las actividades previas, $2, 3, .., i-1$, para darle a la actividad i más posibilidades de encontrar un tiempo de finalización que satisfaga las restricciones. Si el mecanismo tiene éxito, el proceso vuelve a la primera fase y considera el tiempo siguiente s_c^{i+1}. Si fracasa, descartamos la solución combinada.

5.5 Actualización del Conjunto de Referencia

La soluciones combinadas que son inicialmente posibles y las obtenidas por el proceso de reparación pasan a la fase de mejora descrita en la Sección 4.2. Entonces estudiamos las soluciones mejoradas para su posible inclusión en

el Conjunto de Referencia. El Conjunto de Referencia S es actualizado siguiendo el criterio de calidad, es decir, formarán parte de él las mejores soluciones b entre las que forman S y las que provienen de la mejora. Si no se puede actualizar S porque ninguna de las nuevas soluciones es mejor que las existentes entonces el algoritmo se para a menos que se incluya la regeneración de S.

5.6 Regeneración del Conjunto de Referencia

La regeneración del Conjunto de Referencia S tiene dos objetivos. Por un lado introducir diversidad en el conjunto ya que, por la forma en que actualizamos S, muchas soluciones con altos *makespans* pueden ser rápidamente sustituidas por otras que los tengan inferiores pero que sean muy similares a las soluciones que ya hay en S. Por otro, tratamos de obtener soluciones de alta calidad, incluso mejores que las que ya tenemos en S.

Obtenemos las nuevas soluciones aplicando de nuevo el algoritmo del GRASP descrito en la Sección 4, pero con una modificación. Sacamos provecho de la información obtenida hasta este momento sobre la solución óptima para orientar la búsqueda hacia las soluciones de alta calidad. Más en concreto, si la mejor solución conocida tiene un *makespan* s^n_{best}, ponemos el horizonte de planificación $T = s^n_{best}$ y corremos el preproceso de nuevo para reducir los TPF de las actividades. Cuando ahora corremos el GRASP es más difícil obtener soluciones, porque sólo permitimos soluciones con iguales o mejores que s^n_{best}, pero si tenemos éxito conseguimos soluciones de alta calidad.

6 Resultados computacionales

6.1 Instancias para las pruebas

Böttcher et al.[3] generaron un primer conjunto de instancias para pruebas. Tomaron como punto de partida PROGEN 2 [6], un generador de instancias para el clásico RCPSP con recursos renovables, modificaron y agrandaron el conjunto de parámetros y generaron un conjunto de 2160 instances con 10 actividades no ficticias, 10 réplicas para cada una de las 216 combinaciones de valores de los parámetros. Como la mayoría de los problemas fueron imposibles restringieron los valores de los parámetros a las 25 combinaciones más prometedoras y generaron 250 instancias con 15, 20, 25 y 30 actividades operativas, manteniendo siempre la cantidad de 30 recursos.

Más recientemente, Schirmer[12] desarrolló PROGEN 3, una extensión de PROGEN 2, y generó algunas instancias nuevas. En concreto, generó 960 instancias de 10, 20 30 y 40 actividades, con 30 recursos. La mayoría de ellas tenían solución posible, unos pocos de ellos eran imposibles y algunos fueron considerados como *no decididos* porque no se consiguió una solución posible en un tiempo

limitado del algoritmo de *branch and bound* de Böttcher et al.[3]. La Tabla 1 muestra la categoría de los problemas de Schirmer como viene expuesta en [12].

Conjunto de Instancias	No resueltos optimamente	Resueltos optimamente	Problemas posibles	No decididos	Problemas imposibles	Total
J10	39	901	940	11	9	960
J20	203	734	937	23	0	960
J30	181	757	938	22	0	960
J40	183	743	926	34	0	960
Total	606	3135	3741	90	9	3840

Tabla 1: *Problemas para pruebas generados por Schirmer*

Por último, consideramos que era necesario disponer de instancias de mayores dimensiones y utilizamos el PROGEN 3 con los mismos parámetros de Schirmer para generar 960 nuevas instancias de 60 actividades y 30 recursos.

6.2 Resultados del preproceso

Los procedimientos del preproceso indicados en la sección 3 se han aplicado a los problemas de Böttcher et al.[3] de 10, 15, 20, 25 y 30 actividades que estaban disponibles solicitándolos a los autores. En las Tablas 2, 3 y 4 aparecen diferentes aspectos de los resultados. La Tabla 2 muestra los logros del preproceso para determinar el estatus de los problemas. Hemos señalado como *posibles* aquellas instancias para las que el preproceso encuentra una solución y como *imposibles* aquellas que el preproceso puede demostrar que no tienen solución. Cuando no ha sido posible determinar dicho estatus los hemos marcado como *no decididos*. En resumen, podemos decir que nuestro preproceso es muy eficiente para determinar el estatus de una instancia dada.

	n=10	n=15	n=20	n=25	n=30
Problemas	2160	250	250	250	250
Detectados como imposibles	1205	16	17	12	8
Detectados como possibles	879	233	231	236	239
No decididos	76	1	2	2	3
Categoría real	Imposibles	Posible	No decididos	Imposibles	No decididos

Tabla 2: *Problemas de Böttcher et al. - Determinación del estatus*

La última línea de la Tabla 2 muestra el estatus que hemos sido capaces de determinar para los problemas que quedaban por decidir después del preproceso. Para ello usamos CPLEX con una formulación entera del problema adaptada a

partir de la expuesta en la Sección 2. Incluso en este caso para 2 instancias de tamaño 20 y 3 de tamaño 30 ha sido imposible determinar su estatus, aunque suponemos que serán imposibles.

La Tabla 3 muestra las soluciones óptimas que obtiene el preproceso. En esta tabla no se han incluido los problemas que quedaban como no decididos porque es muy improbable que tengan solución. Para más del 70 % de las instancias las técnicas del preproceso han obtenido los óptimos.

	n=10	n=15	n=20	n=25	n=30
Problemas	2160	250	250	250	250
Problemas posibles	879	234	231	236	239
ptimos obtenidos por el preproceso	646	165	177	190	193
Problemas restantes	233	67	54	46	46

Tabla 3: *Problemas de Böttcher et al. - Soluciones óptimas identificadas en el preproceso*

La Tabla 4 presenta la reducción de recursos y variables conseguida para aquellos problemas no resueltos por el preproceso. Las rápidas técnicas del preproceso reducen significativamente la cantidad de recursos a considerar y, todavía más importante, la cantidad de valores posibles de las variables de decisión.

	n=10	n=15	n=20	n=25	n=30
Problemas	233	67	54	46	46
Recursos iniciales	30	30	30	30	30
Recursos restantes (media)	18 (60%)	18 (60%)	23 (77%)	25 (83%)	25 (83%)
Variables iniciales (media)	90	268	565	874	1314
Variables restantes (media)	51 (57%)	130 (49%)	348 (62%)	611 (70%)	906 (69%)

Tabla 4: *Problemas de Böttcher et al. - Reducción de recursos y variables*

Los resultados obtenidos para los problemas de prueba generados por Schirmer[12] son de calidad similar y pueden ser consultados en [1, 2].

6.3 Resultados computacionales de los algoritmos constructivos

Las 32 reglas de prioridad descritas por Schirmer[12] fueron introducidas en el algoritmo constructivo de la Sección 4.1. Fueron comprobadas con las 879 instancias posibles de tamaño 10 generadas por Böttcher et al.[3]. La Tabla 5 nos muestra los resultados obtenidos por la 6 reglas que mejor han funcionado. Las 3 primeras reglas se basan en el grafo de los problemas, mientras que las 3

últimas lo hacen en el consumo de recursos. ES indica que las reglas requieren solo el uso de recursos escasos con subíndices r. Rk_{rs} es la capacidad restante del recurso r en el estadio s, según se define en la sección 4.1. RD_{jrt} es la demanda relevante, definida como $RD_{jrt} = k_{jr}|Q_{jt} \cap \Pi_r|$. MDE_{jrt} es la demanda mínima relevante comprometida para el recurso r por todos los sucesores de la actividad j cuando esta empieza en el periodo t. La característica más importante de la Tabla 5 es que incluso las mejores reglas son incapaces de obtener una solución posible para el 20% de esas pequeñas instancias de tamaño 10. Por tanto, necesitamos estrategias de aleatorización y mecanismos de reparación que incrementen significativamente la probablidad de encontrar soluciones posibles en la fase constructiva del algoritmo GRASP.

Regla	Definición	Soluciones posibles (%)	Soluciones óptimas (%)
LFT	$Min\{LFT_j\}$	80.09	64.28
MTS	$Max\{\|\{i\|j \in P_i^{'}\}\|\}$	79.64	69.98
SLK	$Min\{LST_j - EFT_j\}$	76.22	61.66
DRC/ES	$Max\{\sum_r (RK_{rs} - RD_{jrt})\}$	81.57	27.08
DRS/ES	$Min\{\sum_r (RK_{rs}/RD_{jrt})\}$	79.29	27.53
TRS/ES	$Min\{\sum_r (RD_{jrt} + MDE_{jrt})\}$	79.41	28.56

Tabla 5: *Resultados de la Reglas Prioridad*

La Tabla 6 muestra los resultados finales del algoritmo constructivo completo con la regla de prioridad LFT, incluyendo ya el mecanismo de reparación. Como en la Tabla 5, los problemas de prueba son los de tamaño 10 de Böttcher et al.[3].

Regla	Estrategia	Iteraciones	Soluciones Posibles (%)	Soluciones ptimas (%)
LFT	Aleatorización 2	1000	99.89	99.09
	Aleatorización 2	2000	100	99.43

Tabla 6: *Resultados del algoritmo constructivo completo*

6.4 Resultados computacionales de los algoritmos GRASP

La Tabla 3 muestra los resultados del algoritmo GRASP en los problemas no triviales de Schirmer[12]. Hemos probado cuatro versiones del algoritmo: *GRASP*, que es el algoritmo GRASP básico, *GR+PR*, en el cual se se utilizan las mejores soluciones obtenidas con el GRASP para realizar el Reencadenamiento de Trayectorias descrito en la Sección 4.4, el *AG-GR*, el procedimiento GRASP modificado descrito en la Sección 4.3, y *AG-GR+PR*, que combina los dos anteriores. El algoritmo GRASP usa la regla de prioridad LFT, el segundo procedimiento de

aleatorización con $\delta = 0.85$ y realiza un máximo de 2000 iteraciones. Para cada tamaño de problema las tablas muestran el número de soluciones no óptimas, la distancia media al óptimo y la distancia máxima al mismo. Sin embargo, no se conocen todas las soluciones óptimas. De hecho, en la Tabla 3, la solución óptima no es conocida para una instancia de tamaño 30 y para 5 de tamaño 40. En estos casos, que están marcados (*), la comparación se realiza con la mejor solución conocida, obtenida mediante un proceso de tiempo limitado realizado con el código entero CPLEX, proveniente de alguna versión del algoritmo GRASP en cualquiera de los tests preliminares o del algoritmo de Scatter Search desarrollado por los autores [2].

Tamaño Problema	Instancias no triviales		*GRASP*	*GR +PR*	*AG − GR*	*AG − GR +PR*
10	803	No óptimos	1	1	1	1
		Dist. Media (%)	0.004	0.004	0.005	0.007
		Dist. Máx. (%)	2.9	2.9	4.0	4.0
		T. Medio CPU (seg.)	0.9	0.9	0.3	0.3
20	565	No óptimos	43	32	22	19
		Dist. Media (%)	0.40	0.33	0.13	0.12
		Max dist. (%)	20.5	20.5	13.0	13.0
		T. Medio CPU (seg.)	1.4	1.4	1.1	1.2
30	453	*No óptimos**	68	63	35	33
		Dist. Media (%)	1.00	0.88	0.24	0.21
		Max dist. (%)	36.4	33.3	13.6	12.1
		T. Medio CPU (seg.)	2.9	3.1	3.4	3.7
40	386	*No óptimos**	89	84	59	54
		Dist. Media (%)	2.03	1.82	0.67	0.59
		Max dist. (%)	38.9	38.9	20.5	20.5
		T. Medio CPU (seg.)	5.7	6.2	2.9	7.2
60	346	No óptimos	110	105	91	80
		Dist. Media a la cota (%)	3.68	3.31	1.38	1.16
		Dist. Máx. a la cota (%)	78.0	78.0	26.8	26.8
		T. Medio CPU (seg.)	8.7	10.3	10.6	13.4

Tabla 7: *Resultados de los algoritmos GRASP en los problemas de Schirmer y en los de 60 actividades*

Los resultados de la Tabla 3 nos permiten observar más claramente las diferencias de resultados entre los cuatro algoritmos. El GRASP agresivo no garantiza una solución mejor que el GRASP básico, pero para los problemas grandes tiende a producir resultados mejores. El algoritmo de Rencadenamiento de Trayectorias añade pequeñas mejoras a los ya buenos resultados obtenidos por los procedimientos GRASP.

Las últimas líneas de cada bloque de la Tabla 3 proporcionan los tiempos

(en segundos de CPU) usados por los algoritmos en los diferentes tamaños de problemas. En todos los casos el preproceso está incluido como un parte del procedimiento de solución. El algoritmo está codificado en *C++* y corre en un Pentium IV a 2.8 Ghz y un mayor detalle sobre la valoración de los tiempos puede encontrarse en [1]. De dicha valoración deducimos que el algoritmo GRASP agresivo con el Reencadenamiento de Trayectorias parece ser la mejor opción para un algoritmo heurístico eficiente.

El último bloque de la Tabla 3 muestra los resultados de los algoritmos GRASP en las instacias de 60 actividades que generamos aleatoriamente. Como las soluciones óptimas no son conocidas, para aquellos problemas cuya optimalidad no hemos podido probar, hemos calculado las distancias entre la mejor solución obtenida y la cota lineal como $(best - lb)/lb$. En este conjunto de instancias el GRASP agresivo es claramente mejor y los tiempos de cálculo no han aumentado mucho.

6.5 Resultados del algoritmo Scatter Search

Para obtener la población inicial se ejecuta el algoritmo GRASP hasta que ha obtenido 100 soluciones posibles o ha llegado a 2000 iteraciones. Con la población inicial se construye un conjunto de referencia S con 10 soluciones de las cuales 5 son soluciones de calidad y 5 lo son de diversidad.

En la Tabla 8 aparecen los resultados de los 2 métodos de combinación descritos en la Sección 5.4. Se indica también la cantidad de soluciones no óptimas para cada método. En esta experiencia preliminar no se incluye ninguna regeneración del conjunto de referencia y el algoritmo se para cuando, después de una fase de combinación, no se han encontrado nuevas soluciones para añadir al conjunto de referencia.

Método de	Soluciones no óptimas				
Combinación	n=10	n=20	n=30	n=40	Total
1	3	25	39	61	128
8	3	22	41	61	127

Tabla 8: *Comparación de los métodos de combinación en los problemas de Schirmer*

La Tabla 8 también muestra que el algoritmo básico de Scatter Search es muy eficiente y obtiene soluciones óptimas para la mayoría de las 3826 instancias posibles de Schirmer. Por tanto, una fase adicional para regenerar el conjunto de referencia sólo estará justificada si ayuda a resolver los problemas más difíciles. El procedimiento de regeneración descrito en la Sección 5.6 depende de tres parámetros: la cantidad de iteraciones del algoritmo GRASP modificado, la cantidad de nuevas soluciones obtenidas y la cantidad de veces que se llama

al proceso de regeneración. Hemos considerado los siguientes valores para esos parámetros:

1. Máxima cantidad de iteraciones: 500 - 1000
2. Máxima cantidad de nuevas soluciones: 20 - 50
3. Llamadas a regenerar: 3 - Sólo cuando la solución ha mejorado desde la última llamada a la regeneración.

Mét. Comb.	Reg	Iter.	Soluciones	Regeneración	Sol. No óptimas	D. Media al óptimo	D. Máx. al óptimo	Tiempo medio (segs)
1	0			Sin regenerar	128	3.19	18.06	35.1
1	1	500	20	Mientras mejore	100	2.17	15.22	66.6
1	2	1000	20	Mientras mejore	94	2.02	15.22	84.5
1	3	1000	50	Mientras mejore	90	1.84	15.22	96.9
1	4	500	20	3 veces	66	1.39	13.04	129.5
1	5	1000	20	3 veces	62	1.30	13.04	172.7
1	6	1000	50	3 veces	60	1.23	15.22	191.3
8	0			Sin regenerar	127	3.30	18.06	47.3
8	1	500	20	Mientras mejore	80	1.78	13.04	86.0
8	2	1000	20	Mientras mejore	76	1.66	13.04	111.0
8	3	1000	50	Mientras mejore	73	1.55	13.04	119.6
8	4	500	20	3 veces	62	1.32	13.04	158.4
8	5	1000	20	3 veces	58	1.14	13.04	210.6
8	6	1000	50	3 veces	53	1.10	13.04	225.3

Tabla 9: *Comparación de los procedimientos de regeneración en los problemas difíciles de Schirmer*

Hemos probado 6 combinaciones de estos parámetros sobre los problemas de Schirmer no resueltos por el algoritmo básico con los diversos métodos de combinación. La Tabla 9 muestra los resultados de los métodos de combinación 1 y 8 sobre los 148 problemas restantes. Para cada estrategia de regeneración las tablas presentan la media y el máximo porcentaje de distancia al óptimo o a las mejores soluciones conocidas porque la solución óptima no es conocida para 1 instancia de tamaño 30 y para 5 de tamaño 40. La última columna proporciona el tiempo medio, en segundos, en un Pentium IV a 2.8 GHz.

Si comparamos línea a línea las dos partes de la tabla, correspondientes respectivamente a los métodos de combinación 1 y 8, podemos ver que las estrategias de regeneración funcionan mejor, en términos de soluciones no óptimas y distancia al óptimo, cuando se usa el método 8 aunque necesita más tiempo de cálculo debido a que hay que combinar 3 soluciones en cada paso. En la parte inferior de la tabla, correspondiente al método 8, podemos ver que la última alternativa, Reg 6, obtiene muy buenos resultados y resuelve óptimamente más del 50% de los problemas restantes. Sin embargo, requiere más tiempo y, como el número de

llamadas a regenerar es fijo, regenerará el conjunto de referencia 3 veces incluso para instancias para las que el algoritmo básico ya habría encontrado la solución óptima. Por tanto, para la experiencia computacional final, reflejada en la tabla 10, también mantenemos la alternativa Reg 1, donde se llama a la regeneración sólo mientras la solución mejora, y también restringe el tiempo máximo de cálculo a 300 segundos por instancia. Este tiempo límite, que se comprueba únicamente al final de cada fase, producirá un ligero deterioro en los resultados globales pero los tiempos medios se reducirán mucho ya que se impiden los tiempos extremadamente largos de algunas instancias.

Tamaño del problema	Instancias	Scatter Search			GRASP
		Regen 0	Regen 1	Regen 6	
		Desviación media respecto al óptimo			
10	946	0.02	0.00	0.00	0.01
20	960	0.09	0.03	0.02	0.07
30	960	0.15	0.09	0.05	0.11
40	960	0.24	0.14	0.10	0.23
Total	3826	0.13	0.07	0.04	0.11
		Soluciones No óptimas			
10	946	3	0	0	1
20	960	22	10	5	19
30	960	41	29	21	33
40	960	61	41	31	54
Total	3826	127	80	57	107
		Tiempo medio de cálculo			
10	946	1.1	1.6	2.1	1.0
20	960	1.8	3.4	10.1	0.7
30	960	3.5	5.7	13.8	2.1
40	960	6.6	10.6	20.7	4.4
Total	3826	3.3	5.3	11.7	2.1

Tabla 10: *Comparación del Scatter Search y el GRASP en los problemas de Schirmer*

La tabla 10 muestra los resultados completos para las tres versiones del algoritmo de Scatter Search: Reg 0, Reg 1, Reg 6, y las compara con la mejor versión del GRASP desarrollado por Alvarez-Valdes et al.[1], el que llamamos Grasp Agresivo con Reencadenamiento de Trayectorias.

Dicha tabla contiene los resultados de las 3826 instancias posibles de Schirmer. La primera parte de la tabla muestra la distancia media a los óptimos (o a las mejores soluciones conocidas, ya que el óptimo es desconocido para 6 instancias de Schirmer). La segunda parte muestra la cantidad de veces que la mejor solución obtenida no consigue el óptimo o la mejor solución conocida, mientras que la tercera parte nos dice el tiempo medio de cálculo. También se puede ver

que mientras que el algoritmo GRASP es muy eficiente y puede obtener mejores soluciones en los problemas pequeños, el Scatter Search con el incremento de complejidad que suponen las estrategias de regeneración puede mejorar significativamente los resultados generales con incrementos moderados de los tiempos de cálculo.

7 Conclusiones

Hemos estudiado una generalización del problema clásico de secuenciación de proyectos con limitación de recursos. Hemos considerado un tipo de recursos relativamente nuevo, los recursos parcialmente renovables, donde la disponibilidad de un recursos está asociada a un conjunto dado de periodos y las actividades sólo lo consumen si son procesadas en dichos periodos. Este tipo de recursos pueden considerarse como una generalización tanto de lo recursos renovables como de los no renovables, pero su principal interés proviene de su utilidad para modelizar situaciones que aparecen en los problemas de turnos, horarios de trabajo y calendarios que pueden ser formulados como problemas de secuenciación de proyectos.

Hemos desarrollado diferentes técnicas de preproceso que ayudan a determinar la existencia de soluciones posibles y a reducir la cantidad de variables y restricciones. También hemos diseñado e implementado unos algoritmos heurísticos basados, por un lado, en el GRASP y el Reencadenamiento de Trayectorias y por otro, en el Scatter Search. El preproceso y los algoritmos heurísticos, que han sido probados en dos conjuntos de instancias provinientes de la literatura y en otro de instancias de mayor tamaño creado por nosotros con un procedimiento similar al utilizado en los otros dos, han sido capaces de determinar el status de factibilidad de muchas de las instancias que hasta el momento estaba sin determinar y de resolver óptimamente la mayoría de las instancias posibles.

Estamos convencidos que las técnicas de preproceso aquí desarrolladas deberían ser usadas por cualquier procedimiento de solución, exacto o heurístico, aplicado a este problema. Nuestros algoritmos heurísticos son muy eficientes y pueden ser consideraros una herramienta útil para obtener soluciones de alta calidad para este problema.

Las líneas futuras de investigación serán el desarrollo de un algoritmo exacto y el diseño de nuevos algoritmos heurísticos para problemas que combinen los recursos parcialmente renovables con los clásicos recursos renovables, dado que esto sucede en muchas situaciones reales.

8 Bibliografía

[1] R. Alvarez-Valdes, E. Crespo, J.M. Tamarit, F. Villa, GRASP and Path Relinking for Project Scheduling under Partially Renewable Resources, European Journal of Operational Research (2006) en prensa.

[2] R. Alvarez-Valdes, E. Crespo, J.M. Tamarit, F. Villa, A Scatter Search Algorithm for Project Scheduling under Partially Renewable Resources, Journal of Heuristics 12 (2006) 95- 113.

[3] J. Böttcher, A. Drexl, R. Kolisch, F. Salewski, Project Scheduling Under Partially Renewable Resource Constraints, Management Science 45 (1999) 544-559.

[4] E.L. Demeulemeester, W.S. Herroelen, Project Scheduling: A Research Handbook, Kluwer Academic Publishers, Boston, 2002.

[5] P. Festa, M.G.C. Resende, GRASP: An annotated bibliography, in: M.G.C. Resende, P. Hansen (Eds.), Essays and Surveys in Metaheuristics, Kluwer Academic Press, Boston, 2001, pp. 325-367.

[6] R. Kolisch, A. Sprecher, A. Drexl, Characterization and generation of a general class of resource-constrained project scheduling problems, Management Science 41 (1995) 1693-1703.

[7] M. Laguna, R.Marti, Scatter Search, Kluwer Academic Publishers, Boston, 2004.

[8] C. Mellentien, C. Schwindt, N. Trautmann, Scheduling the factory pick-up of new cars, OR Spectrum (2004), in press.

[9] K. Neumann, C. Schwindt, N. Trautmann, Advanced production scheduling for batch plants in process industries, OR Spectrum 24 (2002) 251-279.

[10] K. Neumann, C. Schwindt, N. Trautmann, Scheduling of continuous and discontinuous material flows with intermediate storage restrictions, European Journal of Operational Research (2004), in press.

[11] M.G.C. Resende, C.C. Ribeiro, Greedy Randomized Adaptive Search Procedures, in: F. Glover, G. Kochenberger (Eds.), State-of-the-art Handbook in Meteheuristics, Kluwer Academic Press, Boston, 2001, pp. 219-250.

[12] A. Schirmer, Project Scheduling with Scarce Resources, Verlag Dr. Kovac, Hamburg, 2000.

[13] C. Schwindt, N. Trautmann, Scheduling the production of rolling ingots: industrial context, model and solution method, International Transactions in Operations Research 10 (2000) 547-563.

Búsqueda por Entornos Variables para Planificación Logística*

J.A. Moreno Pérez[a] **y N. Mladenović**[b]

[a]DEIOC. Instituto Universitario de Desarrollo Regional,
Universidad de La Laguna
[b]School of Mathematics,
Brunel University, London

1 Introducción

Las Metaheurísticas son estrategias generales para diseñar procedimientos heurísticos para resolver un problema de optimización mediante un proceso de búsqueda en un cierto espacio de soluciones alternativas. Los procesos de búsqueda heurística están generalmente basados en transformaciones de las alternativas que determinan una estructura de entornos en el espacio de soluciones. La **Búsqueda por Entornos Variables** (Variable Neighbourhood Search **(VNS)** [60], [38], [48] está basada en un principio simple: cambiar sistemáticamente la estructura de entornos por la que se realiza la búsqueda [40], [45]. Su desarrollo ha sido rápido, con muchos artículos ya publicados o pendientes de aparecer [43]. Se han realizado muchas extensiones, principalmente para permitir la solución de problemas de gran tamaño [42]. En la mayoría de ellas, se ha hecho un esfuerzo por mantener la simplicidad del esquema básico [47][46].

En la siguiente sección se exponen las reglas básicas de la VNS. Las extensiones, incluyendo los híbridos, se consideran en la sección 3. En la sección 4 se repasan las aplicaciones prácticas más importantes de la VNS en planificación

*Este trabajo ha sido parcialmente financiado por los proyectos TIN2005–08404–C04–03 (el 70% son fondos FEDER) y PI042005/044

logística. En la sección 5 se profundiza en algunas cuestiones relevantes de esta metaheurística. El trabajo finaliza con unas breves conclusiones.

2 Esquemas Fundamentales

Un problema de optimización consiste en encontrar, dentro de un conjunto X de soluciones factibles, la que optimiza una función $f(x)$. Si el problema es de minimización se formula como sigue:

$$\min\{f(x) \mid x \in X\} \qquad (1)$$

donde x representa una **solución** alternativa, f es la **función objetivo** y X es el **espacio de soluciones** factibles del problema. Una **solución óptima** x^* (o mínimo global) del problema es una solución factible donde se alcanza el mínimo de (1).

Si X es un conjunto finito pero de gran tamaño es un problema de **optimización combinatoria**. Si $X = \mathbb{R}^n$, hablamos de **optimización continua**. La mayoría de los problemas de optimización que surgen en aplicaciones prácticas son **NP**-duros [27] y para abordarlos se necesitan métodos de optimización heurística (al menos para instancias de gran tamaño o como solución inicial para algún procedimiento exacto). Las metaheurísticas se han mostrado como una herramienta apropiada para abordar este tipo de tareas.

Una **estructura de entornos** en el espacio de soluciones X es una aplicación $N : X \rightarrow 2^X$ que asocia a cada solución $x \in X$ un entorno de soluciones $N(x) \subseteq X$, que se dicen *vecinas* de x. Las metaheurísticas de búsqueda local aplican una transformación o movimiento a la solución de búsqueda y por tanto utilizan, explícita o implícitamente, una estructura de entornos. El entorno de una solución $x \in X$ estaría constituido por todas aquellas soluciones que se pueden obtener desde x mediante una de las transformaciones o movimientos contemplados.

Una solución factible $x^* \in X$ es un **mínimo global** del problema (1) si no existe una solución $x \in X$ tal que $f(x) < f(x^*)$. Decimos que la solución $x' \in X$ es un **mínimo local** con respecto a N si no existe una solución $x \in N(x') \subseteq X$ tal que $f(x) < f(x')$. Una búsqueda local descendente cambia la solución actual por otra solución mejor de su entorno, por tanto corren el riesgo de quedarse atascada en un mínimo local que no sea óptimo global. Las metaheurísticas basadas en procedimientos de búsqueda local aplican distintas formas de continuar la búsqueda después de encontrar el primer óptimo local. La metaheurística VNS consiste básicamente en cambiar la estructura de entornos de forma sistemática.

La metaheurística VNS se basa en aprovechar sistemáticamente tres hechos simples:

1. Un mínimo local con una estructura de entornos no lo es necesariamente con otra.

2. Un mínimo global es mínimo local con todas las posibles estructuras de entornos.

3. Para muchos problemas, los mínimos locales están relativamente próximos entre sí.

Los dos primeros hechos sugieren el empleo de varias estructuras de entornos en las búsquedas locales para abordar un problema de optimización. El último hecho, constatado empíricamente, indica que los óptimos locales proporcionan información acerca del óptimo global. Puede ser, por ejemplo, que tengan características comunes pero, generalmente, no se sabe cuales son esas características. Es conveniente, por tanto, realizar un análisis de las proximidades de cualquier óptimo local buscando información que permita orientar la búsqueda hacia el óptimo global.

Las heurísticas basadas en VNS, al contrario de lo que ocurre con otras metaheurísticas, se mantienen simples; no sólo sus esquemas básicos sino también la mayoría de las extensiones, requiriendo el ajuste de muy pocos parámetros. Esta característica permite que la metaheurística VNS y sus extensiones sean útiles para diseñar rápidamente procedimientos heurísticos con los que proporcionar buenas soluciones con rapidez de manera muy simple dejando al descubierto cuales son las razones que determinan su rendimiento, lo que frecuentemente facilita la elaboración de implementaciones sofisticadas muy eficientes.

Al abordar la resolución de un problema, o tipo de problemas, particular, una de las primeras tareas consiste en la recopilación de información y documentación acerca del problema y de sus características. Esta información debe incluir aspectos referentes a, aspectos de los problemas reales, su dificultad y los procedimientos, heurísticos o no, aplicados al enfrentarse a ellos. Esta información puede completarse con algunos diseños intuitivos de operaciones de mejora de las soluciones propuestas o información aportada por la experiencia del rendimiento de tales operaciones. Esta información y el conocimiento de procedimientos rudimentarios, generalmente búsquedas locales, puede ser aprovechada para el diseño de ingredientes de la VNS que pueden ser determinantes en su éxito.

En los problemas de logística más importantes, las soluciones están constituidas por selecciones u ordenaciones de elementos. Una forma corriente de realizar movimientos en el espacio de soluciones es cambiar algunos de estos elementos. Estos movimientos dan lugar a diversas estructuras de entornos si se fija algunos de los aspectos de estos cambios. En muchos procedimientos de búsqueda se utilizan estos movimientos fijando o acotando el número de elementos de la solución que se pueden cambiar. De esta forma se obtienen los entornos más utilizados las búsquedas de entornos variables. Un k- intercambio es un movimiento consistente en intercambiar k elementos de la solución. Es decir, cambiar k elementos de la solución por otros k elementos que no estén en la solución, si las soluciones consisten en la selección de un número fijo de elementos, o intercambiar de posición k

elementos de la solución, si se trata de ordenaciones. Se suelen utilizar los entornos $N_k(x)$ consistentes en cambiar k elementos en la solución x, para $k = 1, 2, ...$ En otras aplicaciones el k-ésimo entorno consiste en las soluciones que se obtienen al cambiar, a lo sumo, k elementos de la solución. Este es el tipo de entornos más corriente en las aplicaciones de la VNS a problemas de logísticos estándares como el problema de la p-mediana [38]. Para los problemas de rutas de vehículos, como el problema del vendedor o TSP (*Travelling Salesman Problem*) también se utilizan otros movimientos estándares en las heurísticas de estos problemas [62].

2.1 VNS Descendente

Una búsqueda local descendente consiste básicamente en determinar iterativamente una mejor solución a partir de la solución actual mediante alguna transformación o movimiento. La clásica búsqueda descendente *greedy* o **voraz** consiste en reemplazar siempre la solución actual por la mejor de todas las soluciones que se pueden obtener a partir de la actual mediante uno de los movimientos contemplados. En el extremo opuesto a la estrategia voraz, de "el mejor primero", se encuentra la estrategia **ansiosa**, de "el primero mejor", que aplica un movimiento de mejora desde que se detecte alguno. Otras estrategias intermedias son posibles para elegir esta solución del entorno que mejora la solución actual, pero todas ellas deben detenerse cuando no sea posible encontrar dicha mejora. Seguramente la elección de los movimientos a considerar puede ser determinante en el éxito de la búsqueda local, pero difícilmente se puede determinar a priori cual de las posibilidades vislumbradas va a ser la más efectiva. Frente a las estrategias de probarlas una de tras de otra en cada caso o emplearlas conjuntamente contemplando todos los movimientos posibles, que son estrategias poco inteligentes, el hecho 1 reseñado anteriormente indica la posible conveniencia de combinar los distintos tipos de movimientos.

Si en una búsqueda local descendente se realiza un cambio de estructura de entornos cada vez que se llega a un mínimo local, se obtiene la **Búsqueda por Entornos Variables Descendente** (*Variable Neighbourhood Descent*, VND). Denotemos por N_k, $k = 1, ..., k_{max}$, a una serie finita de estructuras de entornos en el espacio X. Los entornos N_k pueden ser inducidos por una o más métricas introducidas en el espacio de soluciones x.

Los pasos de la VND se muestran en la figura 1.

Inicialización. Seleccionar una serie de estructuras de entornos N_k, $k = 1, ..., k_{max}$, y una solución inicial x.

Iteraciones. Repetir, hasta que no se obtenga mejora, la siguiente secuencia:

1. Hacer $k \leftarrow 1$.

2. Repetir, hasta que $k = k_{max}$, los pasos:

   (a) Exploración del entorno. Encontrar la mejor solución x' del k-ésimo entorno de x.

   (b) Moverse o no. Si la solución obtenida x' es mejor que x, hacer $x \leftarrow x'$ y $k \leftarrow 1$; en otro caso, hacer $k \leftarrow k + 1$.

Figura 1. VNS Descendente; VND

Nótese que, dado que se vuelve a iniciar el recorrido de las estructuras de entornos desde la primera cada vez que se produce un cambio, cuando se trata de encontrar la mejor solución del k-ésimo entorno de la solución actual x, esta solución no ha podido ser mejorada con ninguno de los (k-1) entornos anteriores. Por tanto, la solución final proporcionada por el algoritmo es un mínimo local con respecto a cada una de las k_{max} estructuras de entornos. Como consecuencia de ellos, la probabilidad de alcanzar un mínimo global es mayor que usando una sola estructura.

En la selección de la estructura de entornos para abordar un determinado problema, tanto implementando una búsqueda local, voraz o ansiosa, o cualquier otra metaheurística de búsqueda que utilice los entornos de forma explícita o implícita deben tenerse en cuenta diversas cuestiones sobre los entornos. Entre tales cuestiones está la completitud, complejidad y número de los movimientos a aplicar, la eficiencia de los mismos, la posibilidad de examinarlos en distinto orden, e incluso el grado de precisión. Estos aspectos son también cruciales para el uso de varias estructuras de entornos en una VNS.

La mayoría de las heurísticas de búsqueda local usan en sus descensos simplemente un entorno y algunas veces dos ($k_{max} \leq 2$). Además del orden *secuencial* de las estructuras de entornos en la VND anterior, se puede desarrollar una estrategia *anidada*. Supongamos, por ejemplo, que $k_{max} = 3$. Entonces una posible estrategia anidada es: ejecutar la VND de la figura 1 para las dos primeras estructuras de entornos, sobre cada x' que pertenezca al tercer entorno de x ($x' \in N_3(x)$). Esta VNS ha sido aplicada en [6], [44] y [4].

2.2 VNS Reducida

Aparte de la búsqueda local descendente, otro método de búsqueda basado explícitamente o implícitamente en una estructura de entornos y aún más rudimentario pero de uso muy frecuente es la búsqueda por recorrido al azar. Mientras que la búsqueda local concentra su esfuerzo en la explotación o intensificación de la búsqueda entorno a la solución actual, la búsqueda por recorrido al azar, por

el contrario, persigue una mayor capacidad de exploración o diversificación de la búsqueda. El procedimiento consiste en realizar transformaciones de la solución actual hasta encontrar una mejora; cuando esto ocurre, se toma la nueva solución como solución actual y se continúa la búsqueda desde ella. En la elección de estas transformaciones o movimientos es conveniente tener en cuenta también una serie de aspectos: en qué dirección realizar los movimientos, qué lejanía abarcar en las transformaciones. Si se repiten los fracasos en la búsqueda de una solución mejor, se puede probar otra forma distinta de realizar las transformaciones. La VNS reducida realiza estos cambios de forma sistemática.

La **Búsqueda por Entornos Variables Reducida** (*Reduced Variable Neighbourhood Search*, RVNS) selecciona al azar soluciones del entorno actual de la solución actual cambiando a la siguiente estructura de entornos si no se obtiene mejora y volviendo a la primera estructura en otro caso. Los pasos de la RVNS se presentan en la figura 2.

Inicialización Seleccionar una serie de estructuras de entornos N_k, $k = 1$, ..., k_{max}, y una solución inicial x. Elegir una condición de parada

Iteraciones. Repetir, hasta que se cumpla la condición de parada, la siguiente secuencia:

1. Hacer $k \leftarrow 1$.
2. Repetir, hasta que $k = k_{max}$, los pasos:
   (a) Agitación. Generar al azar una solución x' del k- ésimo entorno de x ($x \in N_k(x)$).
   (b) Moverse o no. Si la solución obtenida x' es mejor que x, hacer $x \leftarrow x'$ y $k \leftarrow 1$; en otro caso, hacer $k \leftarrow k + 1$.

Figura 2. VNS Reducida; RVNS.

El cambio de estructura de entornos se puede realizar sólo si la mejora no se produce en cierto número de intentos. En este caso, se considera que varias estructuras de entornos consecutivas coinciden; $N_k = N_{k+1}$, para algunos valores de k. Frecuentemente, la distinta forma de realizar los movimientos que de lugar a las distintas estructuras de entornos puede significar probar soluciones más distantes o más distintas. En estos casos se está usando una serie de estructuras de entornos anidadas, es decir donde $N_k(x) \subseteq N_{k+1}(x)$, y por tanto el cambio de estructura de entornos se interpreta como una ampliación del entorno o del

radio de búsqueda de la mejora. La ampliación del radio se realiza por escalones si $N_k(x) = N_{k+1}(x)$ para una serie de iteraciones consecutivas.

La RVNS es útil para instancias muy grandes de problemas en las que la búsqueda local es muy costosa. Como condición de parada se usa generalmente el máximo número de iteraciones entre dos mejoras. La RVNS se ha mostrado superior a un método de Monte-Carlo en un problema minimax continuo y a la heurística del *intercambio rápido* de Whittaker al aplicarla al problema de la p-mediana [45].

2.3 VNS Básica

La **Búsqueda por Entornos Variables Básica** (*Basic Variable Neighbourhood Search*, BVNS) es una estrategia que alterna búsquedas locales con movimientos aleatorios sobre unas estructuras de entornos que varían de forma sistemática.

Los pasos de la VNS básica se dan en la figura 3.

Inicialización Seleccionar una serie de estructuras de entornos N_k,$k = 1$, ..., k_{max}, y una solución inicial x. Elegir una condición de parada

Iteraciones. Repetir, hasta que se cumpla la condición de parada, la siguiente secuencia:

1. Hacer $k \leftarrow 1$.
2. Repetir, hasta que $k = k_{max}$, los pasos:
   (a) Agitación. Generar al azar una solución x' del k- ésimo entorno de x.
   (b) Moverse o no. Si la solución obtenida x'' es mejor que x, hacer $x \leftarrow x''$ y $k \leftarrow 1$; en otro caso, hacer $k \leftarrow k + 1$.

Figura 3. VNS Básica; BVNS.

Las estructuras de entornos utilizadas pueden ser anidadas y escalonadas como en la búsqueda por entornos variables reducida. La condición de parada puede ser, por ejemplo, el máximo tiempo de CPU permitido, el máximo número de iteraciones, o el máximo número de iteraciones entre dos mejoras. Obsérvese que la solución x' se genera al azar en el paso (2a) para evitar el ciclado prematuro, que puede ocurrir si se usa cualquier regla determinística.

2.4 VNS General

La Búsqueda por **Entornos Variables General** (*General Variable Neighbourhood Search*, GVNS) se obtiene al sustituir la búsqueda local del paso (2b) de la Búsqueda por Entornos Variables básica por una Búsqueda por Entornos Variables descendente; es decir, una VND. Esta estrategia utiliza dos series de estructuras de entornos posiblemente distintas, una para la búsqueda descendente y otra para los movimientos aleatorios de agitación.

Los pasos de la VNS general (GVNS) se muestran en la figura 4.

Inicialización Seleccionar una serie de estructuras de entornos N_k, $k = 1, ..., k_{max}$, que se usarán en la agitación; y una serie de estructuras de entornos N'_j, $j = 1, ..., j_{max}$, que se usarán en el descenso y una solución inicial x. Elegir una condición de parada

Iteraciones. Repetir, hasta que se cumpla la condición de parada, la siguiente secuencia:

1. Hacer $k \leftarrow 1$.
2. Repetir, hasta que $k = k_{max}$, los pasos:
   (a) Agitación. Generar al azar una solución x' del k- ésimo entorno de x.
   (b) Búsqueda local. Aplicar la VND con los entornos N'_j, $j = 1, ..., j_{max}$, y x' como solución inicial; denótese con x'' la solución así obtenida.
   (c) Moverse o no. Si la solución obtenida x'' es mejor que x, hacer $x \leftarrow x''$ y $k \leftarrow 1$; en otro caso, hacer $k \leftarrow k + 1$.

Figura 4. VNS General; GVNS.

El uso de la Búsqueda por Entornos Variables general (GVNS) ha dado lugar a las aplicaciones más exitosas aparecidas recientemente (ver, por ejemplo, [44],[6], [10], [11], [12], [13], [71] y [75]).

2.5 VNS Anidada

Es el caso más sencillo del uso de una VNS general. Ambas series de entorno coinciden en una serie anidada obtenida a partir de un único tipo de movimiento sencillo. El movimiento más usual en este sentido es el del intercambio que consiste en cambiar un elemento de la solución por otro elemento posible. Si las soluciones se identifican como vectores el cambio consiste en modificar una de las

componentes; si se trata de la selección de un número determinado de elementos de un universo, el cambio consiste en sustituir un elemento de la solución por otro que no esté en la solución; si se trata de permutaciones, se intercambian las posiciones de dos elementos. El valor de k corresponde con el número de elementos de la solución cambiados (esta ha sido la versión paralelizada en [25]).

La estrategia de **Búsquedas por Entornos Variables Anidados** (*Nested Variable Neighbourhood Search*, NVNS) se obtiene a partir de un único movimiento elemental al que está asociada una estructura de entornos N. Los entornos anidados se definen recursivamente por $N_1(x) = N(x)$ y $N_{k+1}(x) = N(N_k(x)), \forall x \in X$. El número máximo de entornos a usar en la agitación y en la búsqueda local se fijan por los valores k_{max} y j_{max}, respectivamente.

Los pasos de la VNS anidada (NVNS) se muestran en la figura 5.

Inicialización Seleccionar una estructura de entornos N y una solución inicial x. Elegir una condición de parada

Iteraciones. Repetir, hasta que se cumpla la condición de parada, la siguiente secuencia:

1. Hacer $k \leftarrow 1$.

2. Repetir, hasta que $k = k_{max}$, los pasos:

   (a) Agitación. Generar al azar una solución x' del k- ésimo entorno de x.

   (b) Búsqueda local con VND.

      (i) Hacer $j \leftarrow 1$.

      (ii) Repetir, hasta que $j = j_{max}$, los pasos:

         * Exploración del entorno. Encontrar la mejor solución $x'' \in N_j(x')$.
         * Moverse o no. Si $f(x'')$ ¡ $f(x')$, hacer $x' \leftarrow x''$ y j? ←?1; en otro caso, hacer $j \leftarrow j + 1$.

   (c) Moverse o no. Si la solución obtenida x'' es mejor que x, hacer $x \leftarrow x''$ y $k \leftarrow 1$; en otro caso, hacer $k \leftarrow k + 1$.

Figura 5. VNS anidada; NVNS.

3 Extensiones de la VNS

Se han propuesto en la literatura diversas formas de extender la VNS para dotarla de algunas características adicionales. Describimos en primer lugar varias formas sencillas de realizar estas extensiones en la subsección 3.1. Las tres siguientes subsecciones se dedican a otras tantas extensiones que constituyen mejoras prácticas de la VNS que han permitido resolver con éxito problemas muy grandes: la Búsqueda por Entornos Variables con descomposición (VNDS), la Búsqueda por Entornos Variables sesgada (SVNS) y la Búsqueda por Entornos Variables paralela (PVNS). En la subsección 3.5 se consideran las extensiones por hibridación de la VNS. En algunos casos se han propuesto independientemente procedimientos heurísticos que en esencia explotan ideas del VNS y que pueden considerarse como casos específicos o extensiones de esta metaheurísticas como la búsqueda local iterada (*Iterated Local Search* ,ILS) o la búsqueda por entornos grandes (*Large Neighbourhood Search*, LNS).

3.1 Extensiones básicas

Las primeras extensiones se derivan directamente de la VNS básica. La BVNS es un método descendente de la primera mejora con aleatorización. Sin mucho esfuerzo adicional se transforma en un método ascendente-descendente: en el paso (2c) hacer también $x \leftarrow x''$ con alguna probabilidad, incluso si la solución es peor que la actual (o que la mejor solución encontrada hasta el momento). La búsqueda en los entornos se intensifica si en lugar de una agitación para tomar una única solución al azar se toman en el paso (2a) todas las soluciones del k-ésimo entorno de la solución actual. En este caso es conveniente explotar un posible recorrido sistemático del entorno. También se puede transformar en una búsqueda de la mejor mejora si se realiza la búsqueda local del paso (2b) desde una solución de cada uno de los k_{max} entornos y se aplica el cambio al mejor entorno $k*$. Una estrategia distinta con intensificación intermedia se tiene eligiendo la solución x' en el paso (2a) como la mejor entre b (un parámetro) soluciones generadas aleatoriamente en el k-ésimo entorno. Una última extensión sencilla consiste en introducir k_{min} y k_{paso}, dos parámetros que controlan el proceso de cambio de entorno: en el algoritmo anterior, en vez de $k \leftarrow 1$ hacer $k \leftarrow k_{min}$ y en vez de $k \leftarrow k + 1$ hacer $k \leftarrow k + k_{paso}$. Estas extensiones pueden aplicarse a la vez produciendo nuevas variantes de la búsqueda por entornos variables.

3.2 VNS con Descomposición

La **Búsqueda por Entornos Variables con Descomposición** (*Variable Neighbourhood Decomposition Search*, VNDS) [51] extiende la VNS en un esquema de entornos variables en dos niveles basado en la descomposición del problema. Cuando se obtiene al azar la solución agitada, se fijan los atributos comunes entre

la solución actual y la agitada y se aborda el problema de optimizar los elementos no comunes, mediante un procedimiento exacto u otra heurística. Sus pasos son los presentados en la figura 6.

Inicialización Seleccionar una serie de estructuras de entornos N_k, $k = 1$, ..., k_{max}, y una solución inicial x. Elegir una condición de parada

Iteraciones. Repetir, hasta que se cumpla la condición de parada, la siguiente secuencia:

1. Hacer $k \leftarrow 1$.
2. Repetir, hasta que $k = k_{max}$, los pasos:
   (a) Agitación. Generar al azar una solución x' del k- ésimo entorno de x. Definir y como el conjunto de atributos presentes en x' pero no en x ($y = x' \setminus x$).
   (b) Búsqueda local. Buscar el óptimo parcial en el espacio de los atributos y por alguna heurística. Sea y' la mejor solución encontrada para estos atributos y sea x'' la correspondiente solución del espacio completo obtenida al incorporar a x tales atributos ($x'' = x \setminus x' \cup y'$).
   (c) Moverse o no. Si la solución obtenida x'' es mejor que x, hacer $x \leftarrow x''$ y $k \leftarrow 1$; en otro caso, hacer $k \leftarrow k + 1$.

Figura 6. VNS por Descomposición; VNDS.

La diferencia entre la VNS básica y la VNDS está en el paso (2b): en vez de aplicar algún método de búsqueda local en el espacio completo X (empezando desde $x' \in N_k(x)$), en la VNDS se resuelve en cada iteración un subproblema en un subespacio $X_k \subseteq N_k(x)$ con $x' \in X_k$. Cuando la búsqueda local utilizada en este paso es también una VNS, aparece un esquema VNS en dos niveles.

3.3 VNS Sesgada

Una vez que se ha alcanzado la mejor solución en una gran región es necesario buscar estrategias para alejarse bastante de ella para posibilitar una nueva mejora. La **Búsqueda por Entornos Variables Sesgada** (*Skewed Variable Neighbourhood Search*, SVNS) [37] afronta la exploración de regiones de valles alejados de la solución actual. Si el óptimo local alcanzado está considerablemente alejado del anterior y no es significativamente peor puede ser conveniente llevar hasta allí el proceso de búsqueda. Las soluciones generadas al azar en entornos muy lejanos

pueden diferenciarse substancialmente de la solución actual; por lo que la VNS degenera, en algún sentido, en una heurística de arranque múltiple (en la que se realizan iterativamente descensos desde soluciones generadas al azar). Por tanto, la VNS sesgada incorpora una compensación por la distancia desde la solución actual para evitar este inconveniente. Sus pasos son presentados en la figura 7.

Inicialización Seleccionar una serie de estructuras de entornos N_k, $k = 1$, ..., k_{max}. Encontrar una solución inicial x y su valor objetiva $f(x)$; hacer $x^* \leftarrow x$ y $f^* \leftarrow f(x)$. Elegir una condición de parada y un parámetro α.

Iteraciones. Repetir, hasta que se cumpla la condición de parada, la siguiente secuencia:

1. Hacer k $\leftarrow 1$.
2. Repetir, hasta que $k = k_{max}$, los pasos:
   (a) Agitación. Generar al azar una solución x' del k- ésimo entorno de x.
   (b) Búsqueda local. Aplicar algún método de búsqueda local con x' como solución inicial; denótese con x'' el mínimo local así obtenido.
   (c) Mejora o no. Si $f(x'') < f^*$, hacer $x^* \leftarrow x''$ y $f^* \leftarrow f(x'')$.
   (d) Moverse o no. Si $f(x'') - \alpha \cdot \rho(x, x'') < f(x)$, hacer $x \leftarrow x''$ y $k \leftarrow 1$; en otro caso, hacer $k \leftarrow k + 1$.

Figura 7. VNS sesgada; SVNS.

La SVNS usa una función $\rho(x, x'')$ para medir la distancia entre la solución actual x y el óptimo local encontrado x''. La distancia usada para definir los entornos N_k puede también utilizarse con este propósito. La elección del parámetro αdebe permitir la exploración de valles lejanos a x cuando $f(x'')$ es algo peor que $f(x)$, pero no demasiado (en otro caso siempre se abandonaría la solución x). Con ello se pretende evitar frecuentes trayectorias desde x a una solución relativamente cercana para volver a x. Un buen valor para α tiene que determinarse experimentalmente en cada caso.

3.4 VNS Paralela

Las **úsquedas por Entornos Variables Paralelas** (*Parallel Variable Neighbourhood Search*, PVNS) constituyen la tercera extensión [66]. Se han propuesto en [25] y [17] diversas formas de paralelizar la VNS que han sido aplicadas al problema de la p- mediana. En [25] se analizan tres de ellas: (i) paralelizar la

búsqueda local, (ii) aumentar el número de soluciones obtenidas del entorno actual y realizar búsquedas locales en paralelo desde cada una de ellas y (iii) hacer lo mismo que en (ii) pero actualizando la información sobre la mejor solución encontrada. En todas ellas se utiliza la conocida estrategia maestro-esclavo en la que el procesador maestro, distribuye la información a los procesadores esclavos que una vez realizada su tarea la devuelven al procesador maestro.

En la primera estrategia de paralelización, los procesadores disponibles comparten la tarea de realizar cada búsqueda local. Por ejemplo, en una búsqueda local *greedy* basada en una estructura de entorno, cada procesador busca la mejor solución de una parte del entorno y se selecciona la mejor de ellas. Cuando ninguno de los procesadores consigue mejorar a la solución, la búsqueda local se detiene. En la segunda paralelización, cada procesador toma una solución del entorno de la solución actual y aplica la búsqueda local. Se toma el mejor mínimo local de los encontrados por los procesadores y se compara con la solución actual a la que reemplaza si la mejora. Sin embargo, en la tercera paralelización cada vez que un procesador se detiene en un mínimo local se compara la solución actual y la reemplaza si la mejora. Entonces, mientras un procesador realiza la búsqueda local es posible que otro procesador haya mejorado la solución actual y con esta solución con la que se compara el mínimo encontrado. Se utiliza la conocida estrategia maestro-esclavo en la que el procesador maestro, que almacena la solución actual x, envía a los procesadores soluciones x'_p del entorno de la solución actual x, recibe los mínimos locales x''_p encontrados por éstos que si mejoran la solución actual x en el momento de recibirlos. La estructura de entornos se modifica cuando ninguno de los procesadores ha conseguido mejorar la solución actual. El inconveniente de esta estrategia es que los procesadores quedan ociosos más tiempo que en la anterior. La segunda paralelización, cuyos pasos se muestran en la figura 8, es la que ha dado mejores resultados.

Inicialización Seleccionar una serie de estructuras de entornos N_k, $k = 1, ..., k_{max}$, y una solución inicial x. Elegir una condición de parada

Iteraciones. Repetir, hasta que se cumpla la condición de parada, la siguiente secuencia:

1. Hacer $k \leftarrow 1$.
2. Repetir en paralelo, hasta que $k = k_{max}$, para cada procesador p los pasos:
   (a) Agitación. Generar al azar una solución x'_p del k- ésimo entorno de x.
   (b) Búsqueda local. Aplicar algún método de búsqueda local con x'_p como solución inicial; denótese con x''_p el mínimo local así obtenido.
   (c) Moverse o no. Si la solución obtenida x''_p es mejor que x, hacer $x \leftarrow x''_p$y $k \leftarrow 1$; en otro caso, hacer $k \leftarrow k + 1$.

Figura 8. VNS Paralela; PVNS.

3.5 Híbridos

Dado que el cambio sistemático de estructura de entornos es una herramienta simple y muy potente, otra forma de extender la VNS ha sido incorporarla a otras metaheurísticas. Estas propuestas han dado lugar a diversas metaheurísticas híbridas.

La búsqueda tabú (*Tabu Search*, TS) [30], [31], [32], [34] generalmente usa una estructura de entornos con respecto a la que ejecuta movimientos de ascenso y descenso explotando diferentes tipos de memoria. En principio hay dos maneras de hacer híbridos de VNS y TS: usar algún tipo de memoria para orientar la búsqueda dentro de la VNS o usar la VNS dentro de la TS. En [67], [8], [5] y [53] se proponen híbridos del primer tipo y en [6] y [19] del segundo tipo.

La metaheurística GRASP (*Greedy Randomized Adaptive Search Procedure*) [22] [73] consta de dos fases; en la primera fase se construyen soluciones usando un procedimiento *greedy* aleatorizado y en la segunda, las soluciones se mejoran por alguna búsqueda local o un método enumerativo. Una forma natural de hibridizar la VNS con GRASP es usar la VNS en la segunda fase de GRASP lo que ha sido aplicado en [58], [75], [1], [10], [68], [23].

La búsqueda Multi-arranque (*MultiStart Search*, MS) [56] [57] es una metaheurística clásica que, para evitar el estancamiento de un procedimiento descendente en un óptimo local, sencillamente reinicia la búsqueda desde otra solución. En [4] se propone y analiza una heurística híbrida entre la VNS y la MS consistente en reiniciar la VNS desde otra solución generada al azar del espacio X, cuando se estanca en un mínimo local, por no encontrar ninguna mejora a través de ninguno de los entornos de la solución x.

En combinación con el método Piloto [21] se ha propuesto recientemente [52]. La VNS se combina con operadores genéticos en [19].

3.6 La VNS en planificación Logística

La VNS ha alcanzado éxitos relevantes en campos de aplicación diversos entre los que caben destacar los de unas técnicas de descubrimiento que han permitido descubrir una herramienta informática (el Sistema AutoGraphiX [13], [2]) que ha conseguido probar o refutar algunas conjeturas abiertas en teoría de Grafos. Se ha aplicado a problemas de optimización continua [61] creando un software específico GLOB [20]. También se ha aplicado con éxito a multitud de problemas de optimización combinatoria importantes en la industrial [43] y la economía [26], entre los que se encuentran los problemas más relevantes de planificación logística [78]: problemas de empaquetado, problemas de localización y problemas de rutas.

Los **Problemas de Empaquetado** constituyen una clase de problemas importantes en contextos de logística y distribución. En [24] se usa la VNS básica para una de las versiones básicas: el problema de empaquetado unidimensional (*Bin-Packing Problem*, BPP) en el que se tiene que empaquetar un conjunto de

objetos de diferente tamaño en el menor número de cajas o segmentos de capacidad fija.

Los **Problemas de Localización**, muy relevantes en la planificación logística, y los de agrupamiento o *clustering* (dentro de la clasificación no supervisada) tienen características comunes que los hacen similares al ser abordados por búsquedas metaheurísticas. El problema de la p-mediana es el problema más extensamente estudiado en Teoría de Localización. Consiste en determinar p ubicaciones, entre un conjunto de m localizaciones posibles para un servicio, de forma que se minimice la suma de las distancias de n usuarios a su punto de servicio más cercano. Este problema ha sido abordado con éxito en [38] y [51] por medio de una VNS básica, una VNS reducida y una VNDS, respectivamente. En [26] y [17] se proponen y prueban diversas VNS paralelas para este problema [66]. En [63] se aplica la VNS al problema del p-centro en el que hay que minimizar, en lugar de la suma, el máximo de las distancias de los usuarios a sus respectivos puntos de servicios más próximos. Diversas variantes de la VNS han sido aplicadas con éxito en [6] y [7] al problema múltiple de Weber que es la versión continua del problema de la p-mediana, en el que las p localizaciones pueden elegirse en todo el plano donde también están ubicados los usuarios. En [51] se muestra la aplicación de la RVNS y la VNDS para el problema simple de localización de plantas (*Simple Plant Location Problem*, SPLP) en el que se debe decidir el número p de localizaciones, minimizando la suma del coste total que implica la separación de los usuarios a los puntos de servicio, y el coste de la selección de cada ubicación para el servicio. El problema de asignación cuadrática (*Quadratic Assignment Problem*, QAP) es otro problema relevante que surge en localización y ha sido abordado con la VNS en [15].

Otro tipo de problemas de optimización combinatoria importante, sobre todo en el contexto de la planificación logística, son los **Problemas de Rutas**. Tanto las versiones clásicas del problema del vendedor o viajante de comercio (*Travelling Salesman Problem*, TSP), del problema de rutas de vehículos (*Vehicle Routing Problem*, VRP) y del problema de rutas de arcos (*Arc Routing Problem*, ARP), así como algunas de sus extensiones, han sido abordadas con la VNS. El problema del viajante de comercio o TSP consiste en, dadas n ciudades con las distancias o costes entre ellas, encontrar una ruta de mínimo coste (es decir, una permutación de las ciudades que minimiza la suma de las n distancias entre ciudades adyacentes en la ruta). En [9], [39] y [62] se aplican procedimientos del tipo de la VNS con distinto tipo de movimiento para el TSP. En [28] se aplica una VNDS similar con dos esquemas de descomposición diferentes. En [76] y [68] se usa una VNS para resolver otra importante extensión del TSP denominada *problema del comprador* en el que, dada una partición del conjunto de clientes, hay que visitar al menos uno de cada conjunto de la partición. Un trabajo interesante sobre el TSP con recogida y distribución es [14].

Un problema de ruta de vehículos o VRP consiste en diseñar las rutas desde un

depósito para visitar un conjunto dado de clientes con mínimo coste. Los clientes tienen demandas conocidas y los vehículos tienen capacidad limitada. En [5] y [15] se aplican la VNS y la VND en la resolución de la variante del problema en el que cada cliente puede ser visitado en cierto intervalo de tiempo que se denomina VRPTW (*VRP with Time Windows*) y en [71] se trata un problema múltiple de este tipo. En [18] se explora el uso de una VND para un VRP con recogida. En [35] se considera un problema on-line muy general.

En [67] se propone una combinación de la VNS y la TS que usa varias estructuras de entornos para el problema del ciclo mediano (*Median Cycle Problem*, MCP). En este problema se debe determinar la ruta de menor longitud que recorra parte de las ciudades con una cota superior para la suma de las distancias desde las ciudades no incluidas en el recorrido hasta la ciudad más cercana de la ruta. En [38] se aplica una VNS a una versión del problema del viajante de comercio con clientes de recogida y entrega de mercancías. La ruta tiene que recorrer todos los clientes de recogida antes que los de entrega, partiendo y llegando al depósito. En [29], [41] y [42] se aborda con una VNS de forma exitosa problemas de rutas de arcos en los que las rutas deben recorrer todas las aristas o arcos de un grafo o red.

3.7 Conclusiones

Las heurísticas más tradicionales en optimización realizaban búsquedas locales descendentes por lo que se bloquean con el primer óptimo local encontrado. Las metaheurísticas proporcionaron métodos para escaparse de los óptimos locales de mala calidad. El valor de tales óptimos locales difiere considerablemente del valor del óptimo global y, especialmente si hay muchos óptimos locales, la mejora en la calidad de los nuevos óptimos alcanzados era insuficiente para acercarse suficientemente. Sin embargo, las mejoras introducidas en las estrategias básicas y los recursos computacionales de las implementaciones han permitido obtener procedimientos cada vez rápidos en acercarse con garantías a la solución óptima por lo que el impacto práctico de las metaheurísticas ha sido inmenso.

En contraste con este éxito, el desarrollo teórico de resultados sobre metaheurísticas está más retrasado [49]. Frecuentemente se obtienen buenas heurísticas con algo de inventiva, experiencia e intuición. y con un gran esfuerzo en el ajuste de numerosos parámetros se van mejorando. Pero las razones de porqué unas estrategias funcionan tan bien como lo hacen para algunos problemas y otras no, y estas funcionan mejor que aquellas para otros problemas son desconocidas. La situación es incluso peor con los híbridos y las mejoras computacionales; muchas veces es difícil discernir si el mérito lo tiene una de las componentes o se está obteniendo beneficio de la interacción.

El desarrollo y aplicación de la VNS contribuye a profundizar en algunas cuestiones que influyen en el rendimiento general de las metaheurísticas. Tres de

estas ideas son la topografía de los óptimos locales, el las trayectorias seguidas por los procesos en su acercamiento al óptimo, y la dicotomía entre la mejor o la primera mejora. El estudio de la topografía de los mínimos locales se realiza en términos de una descripción de los *perfiles de montañas y valles* encontrados durante la búsqueda. Cuando se aplica la VNS, el descenso desde una solución x', seleccionada al azar, puede volver al óptimo local x alrededor del que están centrados los entornos actuales, o a otro óptimo local x'' cuyo valor puede ser mejor o no que el de x. Se ha estudiado en [37] la probabilidad de estos tres sucesos como una función de la distancia de x a x' al aplicarla al problema de la máxima satisfabilidad ponderada. Conviene también observar si x'' está más cerca de x que x' (lo que puede ser interpretado como que x'' pertenece al mismo valle que x, con rugosidades locales en relieve) o no (lo que puede interpretarse como un indicio de que se ha encontrado un nuevo gran valle). El relieve suministra esta información en base a una determinada cantidad de descensos desde puntos en entornos sucesivamente más amplios. Los perfiles varían considerablemente con la calidad del óptimo local x. Cuando x es una mala solución es suficiente alejarse un poco para obtener, con alta probabilidad, un óptimo local mejor. Cuando el óptimo local x es bueno, o muy bueno, debe alejarse bastante más para encontrar un nuevo gran valle y, además, la probabilidad de encontrar una solución mejor que la actual es entonces baja. Esto ilustra una debilidad del esquema básico de la VNS que tiende a degenerar en un arranque múltiple cuando la distancia de x a x' se hace grande. El remedio es el proporcionado por el esquema de la VNS sesgada.

Para algunos problemas se dispone de instancias para las que se conoce el óptimo global y se puede analizar la *distancia al blanco* durante el proceso de búsqueda. Es corriente analizar las búsquedas heurística determinando lo frecuente que se alcanza el óptimo global o el promedio de la diferencia relativa entre el valor óptimo del objetivo y el alcanzado por la búsqueda. Sin embargo, se obtiene mucha más información si se consideran las propias soluciones y no sólo su valoración. Una herramienta para poner en práctica esta idea ha sido desarrollada para aplicar una VNS para el problema del vendedor en [28]. Esta herramienta presenta en pantalla la solución óptima para el caso de entrenamiento bajo estudio, la solución actual y la diferencia simétrica entre estas dos soluciones. Esto indica cuanta mejora queda por hacer y donde. Además, una rutina permite también la representación de la diferencia entre la solución actual en una iteración y en la siguiente. Finalmente, como las representaciones de las soluciones para problemas grandes pueden ser difíciles de leer, y muchos problemas, en particular los problemas euclídeos, permiten una descomposición natural, una rutina de enfoque permite la representación de la información mencionada anteriormente para subproblemas seleccionados; es decir, en alguna región del espacio en la que se traza la ruta. Además de usar esta información para guiar la búsqueda, se puede evaluar el trabajo realizado paso a paso por una VNS u otra metaheurística. Se

pueden estudiar variantes en cada uno de esos pasos y recopilar información detallada para sacar conclusiones que pueden no ser evidentes en un rendimiento global de la heurística. Esto puede dar lugar al descubrimiento de fenómenos inesperados y, la profundización proporcionada por su explicación, puede a su vez dar lugar a principios para mejorar las heurísticas.

En muchos trabajos se ha observado que cuando se aplica un paso de una la heurística buscando iterativamente un movimiento de mejora dentro de un conjunto de posibilidades se puede optar por dos tácticas bien diferenciadas: la primera mejora (es decir, el primer movimiento que reduce el valor de la función objetivo) o la mejor mejora, (es decir, el movimiento que más reduce el valor de la función objetivo). Del análisis de las características de los entornos se deduce si al acercarse al óptimo local es conveniente realizar el esfuerzo de recorrer completamente el entorno de la solución actual o en los estadios iniciales de la búsqueda las mejoras encontradas inicialmente son suficientemente significativas para hacer irrelevante el análisis exhaustivo del entrono.

Los criterios de comparación entre las mateheurísticas deben referirse al mayor o menor grado de cumplimiento de las propiedades *deseables* de las metaheurísticas. Tales buenas cualidades son las que propician o garantizan el interés práctico y teórico de las propuestas. En [59] se propone una propuesta de relación de dichas propiedades. Esta relación es similar a las propuestas por otros autores. En [46] se analiza en que medida la VNS en relación a otras metaheurísticas se ajustan a tales propiedades. Algunas de tales propiedades son las siguientes: la simplicidad, la precisión: la coherencia: la efectividad, la eficacia: la eficiencia, la robustez, la interactividad y la innovación

La VNS está basada en un principio simple poco explorado: el cambio sistemático de la estructura de entornos durante la búsqueda. Esta *simplicidad* de la metaheurística contribuye a dotarla de la amplia aplicabilidad que se refleja en la variedad de las aplicaciones aparecidas en la naturaleza. La VNS está dotada de reglas *precisas* que describen la forma de efectuar tales cambios. Todos los pasos de los esquemas básicos y extendidos de la VNS se traducen *coherentemente* de los principios en que se inspira. La *eficacia* de la VNS se sustenta en la probabilidad de encontrar soluciones óptimas para una gran cantidad de problemas en los que ha sido probada superior o equivalente a otras metaheurísticas con la que ha sido comparada. La VNS tiene probada efectividad en la resolución de problemas de varios bancos de prueba con resultados óptimos o muy cercanos a los óptimos, y con tiempo computacional moderado (o al menos razonable). La metaheurística VNS se ha mostrado considerablemente *eficiente* en muchos experimentos al obtener de una forma más rápida resultados mejores o equivalentes a los de otras metaheurísticas. Además, la VNS se muestra *robusta* ya que ha probado su rendimiento en multitud de problemas sin necesidad de realizar un ajuste específico de parámetros al conjunto de entrenamiento. Las metaheurísticas VNS se ha extendido considerablemente al hibridizarse con otros

procedimientos mejorando su efectividad. Cabe destacar además la frecuencia con la que las metaheurísticas incorporan las ideas en las que se basa la VNS para beneficiarse de sus efectos positivos. La *amigabilidad* de los sistemas basados en la VNS cuenta con la ventaja de que los principios básicos en que se basa son fáciles de usar con un número de parámetros muy pequeño, incluso inexistentes en algunos casos. Las posibilidades de la VNS para la *innovación* están ampliamente corroborada con los resultados obtenidos con el programa AutoGraphiX AGX [13] [2] de descubrimiento científico asistido por ordenador que está basado en VNS y del que ya se han publicado gran cantidad de resultados bajo el título común *Variable Neighborhood Search for Extremal Graphs*.

Las líneas futuras tienen que ir por las mejoras necesarias para poder resolver eficientemente problemas difíciles y muy grandes. El tamaño de los problemas abordados se limita en la práctica por las herramientas disponibles para resolverlos más que por la necesidad de los potenciales usuarios de estas herramientas.

4 Bibliografía

[1] A. Andreatta, C. Ribeiro. Heuristics for the phylogeny problem. Journal of Heuristics, 8: 429-447, 2002.

[2] R.K. Ahuja, J.B. Orlin, D. Sharma . Very large-scale neighborhood search, International Transactions in Operations Research, 7: 301– 317. 2000.

[3] M. Aouchiche, J.M. Bonnefoy, A. Fidahoussen, G. Caporossi, P. Hansen, L. Hiesse, J. Lacheré, A. Monhait (2005). Variable Neighborhood Search for Extremal Graphs, 14. The AutoGraphiX 2 System, En: Liberti L., Maculan N. (eds), Global Optimization: from Theory to Implementation, Springer, 2005.

[4] N. Belacel, P. Hansen, N. Mladenović . Fuzzy J-means: a new heuristic for fuzzy clustering. Pattern Recognition, 35(10):2193- 2200. 2002.

[5] O. Bräysy. A reactive variable neighborhood search algorithm for the vehicle routing problem with time windows. INFORMS Journal on Computing, 15, 347–368. 2003.

[6] J. Brimberg, P. Hansen, N. Mladenović, E. Taillard . Improvements and comparison of heuristics for solving the multisource weber problem. Operations Research, 48(3): 444-460. 2000.

[7] J. Brimberg, N. Mladenović . A variable neighborhood algorithm for solving the continuous location-allocation problem. Studies in Locational Analysis, 10:1-12. 1996

[8] E.K. Burke, P. De Causmaecker, S. Petrović, G.V. Berghe. Variable neighborhood search for nurse rostering. In MIC'2001, pages 755-760, Porto. 2001.

[9] E.K. Burke, P. Cowling, R. Keuthen . Effective local and guided variable neighborhood search methods for the asymmetric travelling salesman problem. Lecture Notes in Computer Science, 2037:203-212. 2001.

[10] S. Canuto, M. Resende, C. Ribeiro. Local search with perturbations for the prize-collecting Steiner tree problem in graphs. Networks, 38:50-58. 2001.

[11] G. Caporossi, D. Cvetković, I. Gutman, P. Hansen. Variable neighborhood search for extremal graphs 2: Finding graphs with extremal energy. Journal of Chemical Information and Computer Science, 39:984-996. 1999.

[12] G. Caporossi, I. Gutman, P. Hansen. Variable neighborhood search for extremal graphs 4: Chemical trees with extremal connectivity index. Computers and Chemistry, 23:469-477. 1999.

[13] G. Caporossi, P. Hansen. Variable neighborhood search for extremal graphs 1: The Auto GraphiX system. Discrete Mathematics, 212:29-44. 2000.

[14] F. Carrabs, J.-F. Cordeau, G., Laporte. Variable Neighbourhood Search for the Pickup and Delivery Traveling Salesman Problem with LIFO Loading, to appear in INFORMS Journal on Computing. 2006.

[15] R. Cordone, R.W. Calvo. A heuristic for the vehicle routing problem with time windows. Journal of Heuristics, 7(2):107-129. 2001.

[16] M.C. Costa, F.R. Monclar, M. Zrikem. Variable neighborhood search for the optimization of cable layout problem. In MIC'2001, pages 749- 753, Porto. 2001.

[17] T.G. Crainic, M. Gendreau, P. Hansen, N. Mladenović. Cooperative parallel variable neighbourhood search for the p-median. Journal of Heuristics 10, 293–314. 2004.

[18] J. Crispim, J. Brandao. Reactive tabu search and variable neighborhood descent applied to the vehicle routing problem with backhauls. In MIC'2001, pages 631-636, Porto. 2001.

[19] T. Davidović, P. Hansen, N. Mladenović. Permutation based genetic, tabu and variable Neighborhood search heuristics for multiprocessor scheduling with communication delays. Asia-Pacific Journal of Operational Research 22, 297–326. 2005.

[20] M. Drazić, V. Kovacević-Vujcić, M. Cangalović, N. Mladenović. GLOB - A new VNS-based software for global optimization. In L. Liberti, N. Maculan, (eds.), Global Optimization: from Theory to Implementation, Vol. 84 de Nonconvex Optimization and its Application Series, Springer 2006.

[21] C.W. Duin, S. Voss. The pilot method: A strategy for heuristic repetition with application to the Steiner problem in graphs. Networks, 34:181–191. 1999.

[22] T. Feo, M. Resende. Greedy randomized adaptive search. Journal of Global Optimization, 6:109-133. 1995.

[23] P. Festa, P. Pardalos, M. Resende, C. Ribeiro. GRASP and VNS for max-cut. In MIC'2001, pages 371-376, Porto. 2001.

[24] K. Fleszar, K.S. Hindi. New heuristics for one-dimensional bin- packing. Computers & Operations Research, 29(7):821-839, 2002.

[25] F. García López, M.B. Melián Batista, J.A. Moreno Pérez, J.M. Moreno Vega. The parallel variable neighborhood search for the p-median problem. Journal of Heuristics, 8, 377-390. 2001.

[26] C.G. García, D. Pérez-Brito, V. Campos, R. Martí. Variable Neighborhood Search for the Linear Ordering Problem. Computers and Operations Research, 33:3549-3565. 2006.

[27] M.R. Garey, D.S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, New-York. 1978.

[28] M. Gendreau, P. Hansen, M. Labbé, N. Mladenović. Variable neighborhood search for the traveling salesman problem. (in preparation). 2001.

[29] G. Ghiani, A. Hertz, G. Laporte. Recent algorithmic advances for arc routing problems. Les Cahiers du GERAD, G-2000-40. 2000.

[30] F. Glover. Tabu search - part I. ORSA Journal of Computing, 1:190- 206. 1989.

[31] F. Glover. Tabu search - part II. ORSA Journal of Computing, 2:4- 32. 1990.

[32] F. Glover, M. Laguna. Tabu search. Kluwer. 1997.

[33] F. Glover, G. Kochenberger (eds.). Handbook of Metaheuristics. volume 57 of International Series in Operations Research & Management Science, Kluwer. 2003.

[34] F. Glover, B. Melián. Tabu Search. Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial, 19(2):29-48, 2003.

[35] A. Goel, V. Gruhn. The General Vehicle Routing Problem. EJOR, por aparecer (2006)

[36] P. Hansen, J. Brimberg, D. Urosević, N. Mladenović. Primal-dual variable neighbourhood search for exact solution of the simple plant location problem (in preparation). 2003.

[37] P. Hansen, J. Jaumard, N. Mladenović, A.D. Parreira. Variable neighbourhood search for maximum weight satisfiability problem. Les Cahiers du GERAD, G-2000-62. 2000.

[38] P. Hansen, N. Mladenović. Variable neighborhood search for the p- median. Location Science, 5:207-226. 1997.

[39] P. Hansen, N. Mladenović. First improvement may be better than best improvement: An empirical study. Les Cahiers du GERAD, G-99- 40. 1999.

[40] P. Hansen, N. Mladenović. An introduction to variable neighborhood search. Capítulo 30 en S.Voss, S. Martello, I Osman, C. Roucairol (eds.), Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization, pág. 433-458. Kluwer, 1999.

[41] P. Hansen, N. Mladenović . A separable approximation dynamic programming algorithm for economic dispatch with transmission losses. Les Cahiers du GERAD, G-2000-31. 2000.

[42] P. Hansen, N. Mladenović. Developments of variable neighborhood search. En C. Ribeiro, P. Hansen (eds.), Essays and Surveys in Metaheuristics, pág. 415-440. Kluwer, 2001.

[43] P. Hansen, N. Mladenović. Industrial applications of the variable neighborhood search metaheuristics. En G.Zaccour (editor), Decisions and Control in Management Science, págs 261-274. Kluwer, 2001.

[44] P. Hansen, N. Mladenović. J-means: A new local search heuristic for minimum sum-of-squares clustering. Pattern Recognition, 34 (2):405-413, 2001.

[45] P.Hansen, N.Mladenović. Variable neighborhood search: Principles and applications. European Journal of Operational Research, 130:449- 467. 2001.

[46] P.Hansen, N.Mladenović. Variable neighbourhood search. Chapter 6 en F.W. Glover, G.A. Kochenberger (eds.), Handbook of Metaheuristics, pp. 145-184. Kluwer, 2003.

[47] P.Hansen, N.Mladenović. Variable neighbourhood search. In Panos M. Pardalos, Mauricio G.C. Resende (eds.), Handbook of Applied Optimization. 2002.

[48] P. Hansen, N. Mladenović. A tutorial on variable neighborhood search. Les Cahiers du GERAD, G-2003-46. 2003.

[49] P. Hansen, N. Mladenović, J. Brimberg, Convergence of variable neighbourhood search, Les Cahiers du GERAD G-2002-21. 2002.

[50] P.Hansen, N.Mladenović, J.A. Moreno Pérez. Búsqueda de Entorno Variable. Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial. Número 19. Vol. 2, 77-92. 2003.

[51] P.Hansen, N.Mladenović, D. Pérez Brito. Variable neighborhood decomposition search. Journal of Heuristics, 7(4):335-350. 2001.

[52] H. Höller, B. Melián, S. Voss. Applying the pilot method to improve VNS and GRASP metaheuristics for the design of SDH/WDM networks. EJOR, por aparecer. 2006.

[53] V.Kovacević, M. Cangalović, M. Asić, D. Drazić, L. Ivanovic. Tabu search methodology in global optimization. Computers & Mathematics with Applications, 37:125–133. 1999.

[54] L. Lobjois, M. Lemaitre, G. Verfaille. Large Neighbourhood Search using Constraint Propagation and Greedy Reconstruction for Valued CSP resolution. In 14th European Conference on Artificial Intelligence (ECAI'2000), Berlin, August 2000, 2001. Workshop on Modelling and Solving Problems with Constraints. 2001.

[55] S. Loudin, P. Boizumault. VNS/ LDS + Cp: A hybrid method for constraint optimization in anytime contexts. In MIC'2001, pages 761- 765, Porto. 2001.

[56] R. Martí. Multi-start methods. In Fred Glover, Gary A. Kochenberger, editors, Handbook of Metaheuristics, chapter 12. Kluwer Academic. 2002.

[57] R. Martí, J.M. Moreno Vega. Métodos Multiarranque. Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial, Vol. 19 pp. 49-60. 2003.

[58] S.L. Martins, M.G.C. Resende, C.C. Ribeiro, P. Pardalos. A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy. Journal of Global Optimization, 17:267-283. 2000.

[59] B. Melián Batista, J.A. Moreno Pérez, J.M. Moreno Vega. Metaheurísticas: una visión global. Inteligencia Artificial. Revista Iberamericna de Inteligencia Artificial. 2003. Número 19. Vol. 2, 7– 28. 2003.

[60] N.Mladenović. A variable neighborhood algorithm - a new metaheuristic for combinatorial optimization. In Abstracts of Papers Presented at Optimization Days, pág. 112. 1995.

[61] N. Mladenović, M. Drazić, V. Kovacević-Vujcić, M. Cangalović. General variable neighborhood search for the continuous optimization. Sometido a EJOR. 2006.

[62] N.Mladenović, P.Hansen. Variable neighborhood search. Computers & Operations Reserach, 24:1097-1100. 1997.

[63] N.Mladenović, M.Labbe, P.Hansen. Solving the p-center problem with tabu search and variable neighborhood search. Networks, 48(1) 48-64.. 2003.

[64] N.Mladenović, J.Petrović, V.Kovacević-Vujcić. M. Cangalović. Solving spread spectrum radar polyphase code design problem by tabu search and variable neighborhood search. European Journal of Operational Research, (to appear). 2004.

[65] N.Mladenović, D. Urosevic. Variable neighborhood search for the k- cardinality. In MIC'2001, pages 749-753, Porto, 2001.

[66] J.A. Moreno Pérez, P. Hansen, N. Mladenović. Parallel Variable Neighborhood Search. Capítulo 11 en E. Alba, editor, Parallel Metaheuristics: A New Class of Algorithms , Wiley, 2005.

[67] J.A. Moreno Pérez, J.M. Moreno Vega, I. Rodríguez Martín. Variable neighborhood tabu search and its application to the median cycle problem. European Journal of Operations Research, 151(2):365-378. 2003.

[68] L.S. Ochi, M.B. Silva, L. Drummond. Metaheuristics based on GRASP and VNS for solving traveling purchaser problem. In MIC'2001, pages 489-494. 2001.

[69] G.Pesant, M.Gendreau. A view of local search in constraint programming, principles and practice of constraint programming. Lecture Notes in Computer Science, 1118:353-366. 1996.

[70] G.Pesant, M.Gendreau. A constraint programming framework for local search methods. Journal of Heuristics, 5:255-279, 1999.

[71] M. Polacek, R. Hartí, K. Doerner, M. Reimann. A variable neighbourhood search for the multi depot vehicle routing problem with time windows. Journal of Heuristics 10, 613–627. 2004.

[72] M.G.C. Resende nad R. Werneck. On the implementation of a swap-based local search procedure for the p-median problem. In: Proceedings of the Fifth Workshop on Algorithm Engineering and Experiments (ALENEX'03), R.E. Ladner (ed.), SIAM, Philadelphia, pp. 119-127. 2003.

[73] M.G.C. Resende, J.L. González Velarde. GRASP: Procedimientos de búsquedas miopes aleatorizados y adaptativos. Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial. Número: 19 Páginas: 61 - 76. 2003.

[74] C.C.Ribeiro, M.C.Souza. Variable neighborhood descent for the degree-constrained minimum spanning tree problem. Discrete Applied Mathematics, 118:43-54. 2002.

[75] C.C.Ribeiro, E.Uchoa, R.Werneck. A hybrid GRASP with perturbations for the Steiner problem in graphs. INFORMS Journal of Computing, 14:228-246. 2002.

[76] M.B. Silva, L.Drummond, L.S. Ochi. Variable neighborhood search for the traveling purchaser problem. In 27th Int. Conference on Computational Industrial Engineering, 2000.

[77] E.Taillard, S.Voss. POPMUSIC - partial optimization metaheuristic under special intensification conditions. In C.Ribeiro, P.Hansen, editors, Essays and Surveys in Metaheuristics, pages 613-630. Kluwer. 2001.

[78] Q.-H. Zhao, S. Chen, C.-X. Zang. Model and Algorithm for Inventory & Routing Decision in a Three-Echelon Logistics System. Submitted to EJOR. 2006.

Nuevos movimientos vecinales basados en "Ejection Chains" para el Minmax VRP*

Jesús F. Alegre y Joaquín A. Pacheco
Dpto. Economía Aplicada
Universidad de Burgos

1 Introducción

El problema de rutas de vehículos, VRP, es sin lugar a dudas uno de los problemas de optimización combinatoria más estudiados y tratados en la literatura. Sin entrar en más detalle y de forma muy genérica el problema consiste en diseñar rutas que visiten una serie de de puntos distribuidos geográficamente de forma que se minimice la distancia total recorrida. En cada punto se debe recoger una cantidad conocida de mercancía. Cada punto debe ser visitado una vez (i.e., por un solo vehículo), las rutas deben empezar y acabar en un depósito central y se deben respetar las restricciones de capacidad del vehículo. Sería prolijo enumerar los trabajos importantes y conocidos que han tratado este modelo y sus variantes, generalizaciones o casos particulares: VRPTW (problema de rutas con ventanas de tiempo), PVRP (Periodic VRP), el conocidísimo TSP, MDVRP (MultiDepot VRP), etc.

*Los autores de este trabajo agradecen la ayuda del Ministerio de Educación y Ciencia por la subvención económica para la realización de este trabajo a través del Plan Nacional de I+D, (Proyecto SEJ- 2005 08923/ECON), así como a la Junta de Castilla y León ("Consejeria de Educación" – Project BU008A06).

Cómo es conocido el VRP es un problema NP-Hard, y aunque hay conocidos técnicas para su resolución de forma exacta como Fisher (1994) o Toth and Vigo (2001), es mucho más extensa la literatura sobre técnicas heurísticas. Desde el clásico algoritmo "saving" de Clarke and Wrigth, (1964), el algoritmo "swap" de Gillet and Miller (1974), o el interesante algoritmo de Fisher y Jailkumar (1981), etc, hasta el desarrollo de técnicas metaheurísticas más modernas, como los *Algoritmos Genéticos* (Potvin and Bengio, 1994; Thangiah, Osman and Sun, 1994), *Temple Simulado* (Osman, 1993), *Búsqueda Tabú* (Gendreau et al., 1994;, Rochat and Taillard, 1995; Taillard et al., 1997; Cordeau, Laporte and Mercier, 2001), GRASP, Kontoravdis and Bard (1995) *Búsqueda Local Guiada* (Vondouris and Tsang, 1999), *Colonias de Hormigas* (Gambardella, Taillard, and Agazzi, 1999) o *Variable Neigbourhood Search* (Bräysy, 2003).

Un elemento importante para el buen funcionamiento de muchos de estas técnicas heurísticas y metaheurísticas es la definición de movimientos o *entornos o vecindarios,* que permitan pasar de una solución a otra cercana en el espacio de búsqueda. Estos movimientos vecinales deben tener las siguientes buenas características: cada vecindario debe contener un gran número de soluciones vecinas y estas deben ser fáciles o rápidas de evaluar. De otra forma la exploración del espacio de soluciones puede no ser eficaz. Tradicionalmente en los problemas de rutas de vehículos los movimientos vecinales pueden ser "intrarutas", es decir, modificaciones de cada ruta independiente de las demás (cambio de orden de los puntos de visita de esa ruta), o "entrerutas", es decir, modificaciones que afectan a mas de una ruta (pasar elementos de una ruta a otra, o intercambio de elementos entre rutas). Entre los primero destacan los conocidos intercambios r-óptimos (Lin, 1965; Lin and Kernighan, 1973) y entre los segundos los propuestos por Van Breedam (1995), o Taillard et al. (1997) que generalizan las usadas anteriormente por Gendreau et al. (1994).

Existen trabajos recientes que usan un mecanismo para generar soluciones de forma diferente. Este mecanismo se basa en el método de "Ejection Chains" ("cadenas de expulsiones") concebido por Glover (1992), en el contexto del TSP. Tiene las siguientes características básicas que lo distinguen de los movimientos clásicos: el movimiento de una solución a otra no es simple, es decir no se pasa de una solución a otra vecina como tradicionalmente, sino que antes de realizarse el cambio se genera una cadena de movimientos y la nueva solución se elige entre las que aparecen en esa cadena. La segunda característica es que estas cadenas no operan directamente sobre soluciones sino sobre estructuras similares a una solución que se denominan "estructuras de referencia". Por tanto para generar estas cadenas de movimientos tipo "Ejection Chains" se ha de disponer una serie de reglas para crear estas estructuras desde una solución, para pasar de una estructura de referencia a otra (reglas de transición), y para crear soluciones factibles a partir de una estructura de referencia. Como se señala en Glover (1996), estas cadenas *"pueden contener cantidades exponenciales de soluciones, pero cuyo mejor*

elemento puede identificarse en tiempo polinomial". Implementaciones de Ejection Chains han producido buenos resultados en problemas de rutas de vehículos incluso con complicadas restricciones como en Cao and Glover (1997), Pesch and Glover, (1997), Rego and Roucairol (1996), Rego, (1998a y 2001).

En este trabajo se trata el Minmax VRP, una variante del VRP en el que la función objetivo no es minimizar la distancia total recorrida, sino la duración de la ruta mas larga. Este problema no ha sido muy estudiado en la literatura sin embargo han aparecido instancias reales de este modelo en el contexto del transporte escolar en áreas rurales y que han sido estudiadas en Delgado and Pacheco (2001), Corberán et al. (2002), Delgado (2002) o Pacheco and Martí (2006). En estos trabajos se usaban metaheurísticos basados en Búsqueda Tabú y Búsqueda Dispersa y se usaban movimientos vecinales clásicos o simples. Una característica en estas instancias reales de transporte escolar rural es que cada ruta empieza en el primer punto de visita y no en el origen (es decir, son rutas abiertas).

En este trabajo se propone el diseño y uso de movimientos basados en Ejection Chains adaptados a este problema. Para ello se propone un nuevo tipo de estructura de referencia y se crean reglas para generar las correspondientes Ejection Chains. La eficacia de estos nuevos movimientos se contrastan con instancias reales de transporte escolar. Los resultados obtenidos muestran que este nuevo tipo de movimiento mejora significativamente los movimientos más clásicos usados para este problema en las anteriores referencias.

El trabajo se estructura de la siguiente manera: en las dos siguientes secciones se describen los movimientos vecinales clásicos "intrarutas" y "entrerutas"; en la sección cuarta se describe el funcionamiento de las Ejection Chains, así como algunas de las ejemplos más importantes de este métodos en problemas de rutas; en la 5 se describe el método Ejection Chains propuesto para este problema, y en la sexta se muestran los resultados computacionales.

A continuación se fija la notación de variables y parámetros usada en este trabajo: $\{1, 2, \ldots, n\}$ el conjunto de puntos del problema, donde 1 es el colegio o destino final, y $\{2, \ldots, n\}$ los puntos de recogida de estudiantes; q_i el número de niños que se recogen en cada punto, $i = 2, \ldots, n$; t_{ij} el tiempo de recorrido entre los puntos i y j, $i, j = 1, \ldots, n$; Q capacidad de los vehículos y m el tamaño de la flota. Finalmente se denota por t_{max} la duración de la ruta más larga, es decir, la función objetivo.

2 Movimientos "Intra rutas"

2.1 Intercambios r-óptimos.

Fueron desarrollados por Lin (1965) y Lin y Kernighan (1973) para el TSP simétrico. Consisten en eliminar r arcos y reconectar las r cadenas[1] restantes ($r+1$ si la ruta no es cerrada). Usualmente se emplean 2-intercambios y 3-intercambios.

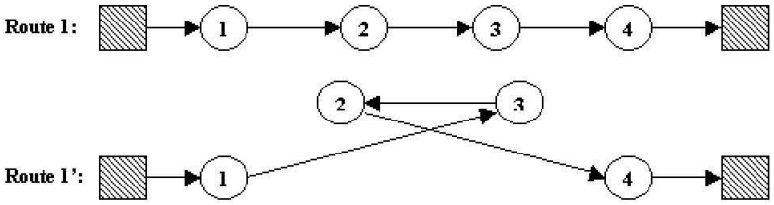


Figura 1.- Intercambio 2-Optimo

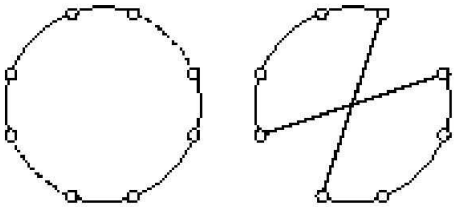


Figura 2.- Intercambio 2-Optimo

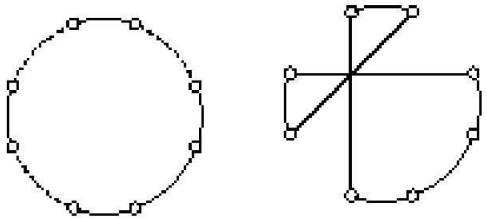


Figura 3.- Intercambio 3-Optimo

Obsérvese que algunos de ellos llevan implícito un cambio de sentido en el recorrido de alguna cadena. Esto puede ser un inconveniente o una ventaja dependiendo del problema concreto.

[1] Cadena: secuencia de puntos consecutivos

2.2 Intercambios de Or

Variante del método anterior, los intercambios de Or (1976) consisten en recolocaciones de cadenas de puntos en la misma ruta. Se suele limitar el tamaño de las cadenas a intercambiar (2 nodos, 3 nodos, etc...). Son por tanto intercambios 3-óptimos. En la figura se recoloca la cadena $[i, i+1]$ después del nodo j.

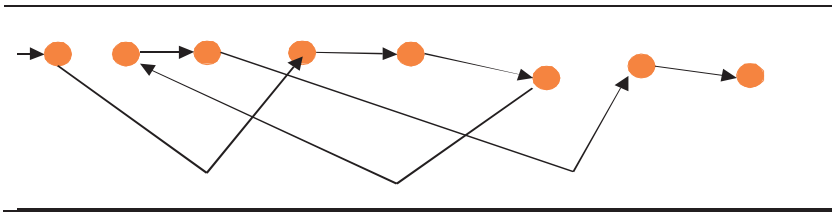


Figura 4.- intercambio de Or (cadena con dos nodos).

2.3 Intercambios IOPT

Variante del método anterior introducido por Braysy, (2003). Son similares a los intercambios de Or, pero en los IOPT intercambios se invierte la orientación de la cadena recolocada.

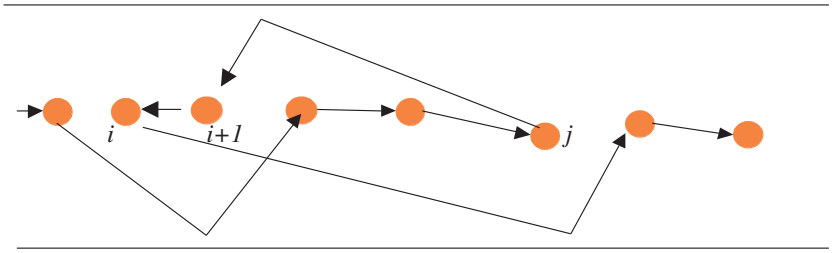


Figura 5.- Intercambio IOPT (cadena con dos nodos).

3 Movimientos "Entre rutas"

3.1 Recolocación de cadenas (String relocation)

Fueron propuestos por Van Breedam (1995). Una cadena de puntos se mueven de una ruta a otra manteniendo el orden original. El caso más simple consiste en mover un punto de una ruta a otra; este tipo de movimientos aparecen en Gendreau et al. (1994).

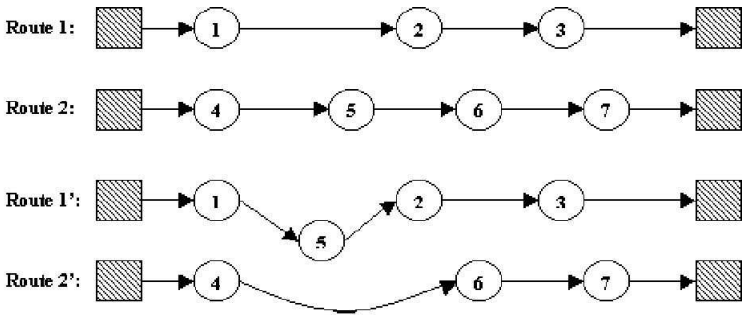


Figura 6.- Movimiento relocate simple

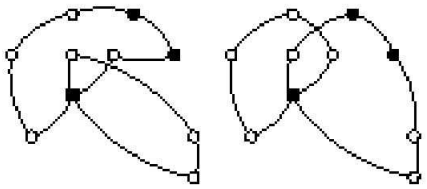


Figura 7: Relocate con 2 nodos

3.2 Intercambio de cadenas (String Exchange)

También propuestos en Van Breedam (1995). De una ruta se envía una cadena de puntos a otra, y de esta última una cadena a la primera. En el caso más simple el movimiento exchange intercambia dos puntos pertenecientes a 2 rutas diferentes; este caso particular aparece también en Gendreau et al. (1994).

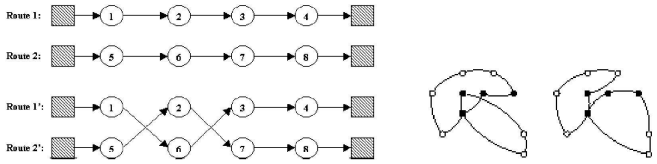


Figura 8.- Movimiento Exchange (caso simple y otro con cadenas de dos y un nodo)

3.3 CROSS intercambios

Propuestos en Taillard et al. (1997). Suponen una generalización de los movimientos entre pares de rutas. Son cambios entre pares de cadenas correspondientes a rutas diferentes.

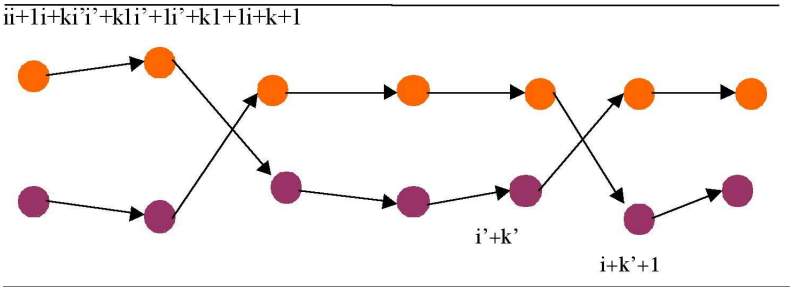


Figura 9.- CROSS Intercambio

Se caracterizan porque mantienen el sentido (dirección) de los clientes en las rutas seleccionadas. Se suele limitar el tamaño de las cadenas a intercambiar. Como ya hemos comentado generalizan los movimientos anteriores: Si $k = k' = 1$, se tiene un *String exchange;* si $k = 1$ y $k' = 0$ un *String relocation;* si $i + k + 1$ e $i' + k' + 1$ coinciden con el final de la ruta se tiene un cruce de rutas

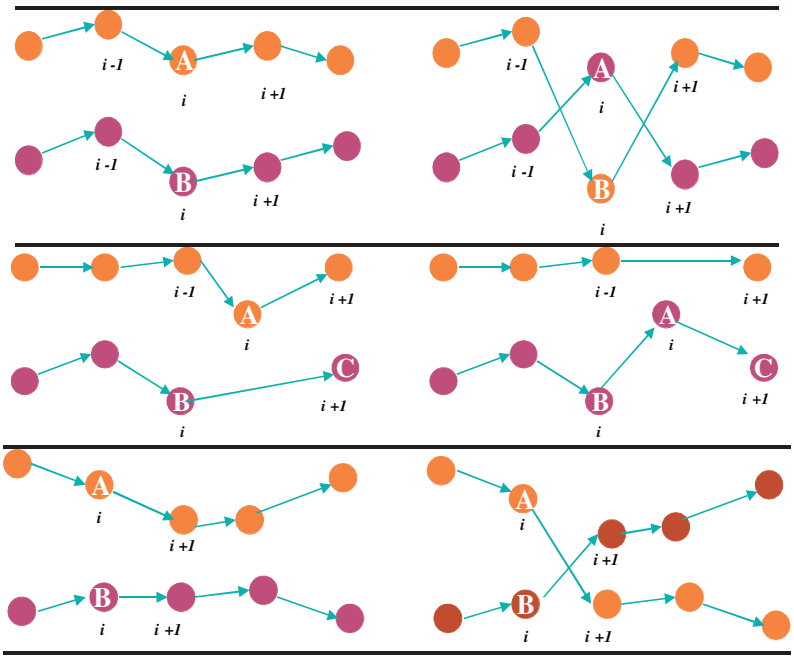


Figura 10.- Casos particulares de CROSS intercambios.

3.4 ICROSS intercambios

Fueron introducidos por Braysy (2003). Son similares a los CROSS intercambio pero en los ICROSS intercambios se invierte la orientación de las cadenas

intercambiadas.

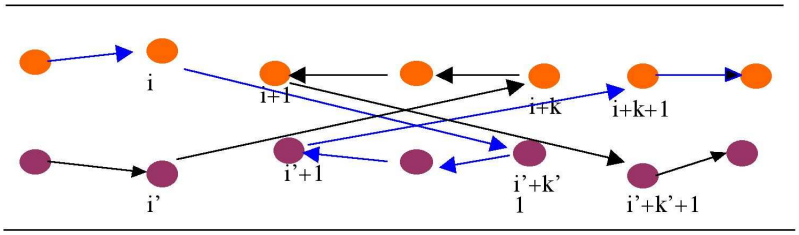


Figura 11.- ICROSS intercambio

4 Concatenación de movimientos simples: "Ejection chains"

Los métodos de *Ejection Chains* fueron concebidos por Glover (1992) en el contexto del TSP para generar movimientos complejos en el espacio de búsqueda a partir de movimientos simples, modificando un número variable de componentes de la solución.

En los métodos de *Ejection Chains* se genera una secuencia de movimientos, que va conduciendo de una solución a otra. En los pasos sucesivos cambios en ciertos elementos causan que otros elementos sean *expulsados* de su estado, posición o valor actual.

Habitualmente las *Ejection Chains* están relacionadas con las restricciones; el término inglés "ejection" significa *expulsión, salida*, y alude a que al hacerse cambios en ciertos elementos se causa que otros elementos sean "expulsados" de su estado actual, debido a que en caso contrario se produciría una infactibilidad. En alguna de las referencias relacionadas con el TSP o el VRP las Ejection Chains se realizan mediante *expulsión* o *salida* de nodos y en otras mediante salida de trayectos o arcos.

Los procedimientos basados en Ejection Chains trabajan de forma explícita sobre estructura llamada *estructura de referencia.* Esta estructura es similar, pero ligeramente diferente de una solución. Por medio de un conjunto de reglas (denominadas "de transición") se va pasando de una estructura a otra. También existen reglas que permiten pasar de una estructura de referencia a soluciones factibles del problema *(soluciones prueba* o *Test Solutions).* Al final la concatenación de movimientos consiste en pasar de una solución de partida a la mejor solución prueba de todas las obtenidas.

Un esquema del funcionamiento de las "Ejection Chains" viene dado el la Figura 12. S indica la solución inicial, ER las estructuras de referencia visitadas y TS las *soluciones pruebas* obtenidas a partir de las estructuras de referencias visitadas.

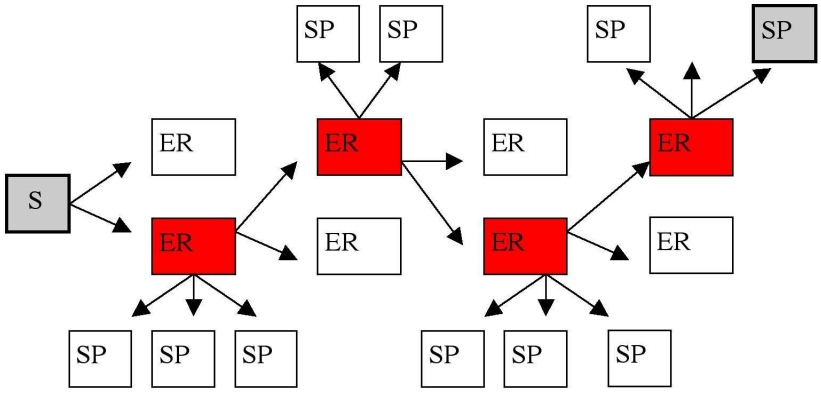


Figura 12.- Esquema de funcionamiento de las "Ejection Chains"

Veamos algunos trabajos en que métodos basados en Ejection Chains han conseguido buenos resultados.

4.1 "Stem-and-cycle", SC

Rego (1998-a), en el contexto del TSP, diseña *Ejection Chains* que se basan en la salida de arcos; para ello utiliza una estructura de referencia, que se representa en la figura 13, llamada "stem-and- cycle" (tallo y ciclo). El tallo va desde t, o punta, hasta r, raíz o nodo común con el ciclo, a los nodos, pertenecientes al ciclo, que están al lado de la raíz se les denomina subraíces ($s1$ y s_2 en la Figura 13). Los nodos t y r tienen un número impar de conexiones. Esta estructura se dice que está degenerada si el tallo se reduce a un solo nodo (se trata entonces de una solución factible).

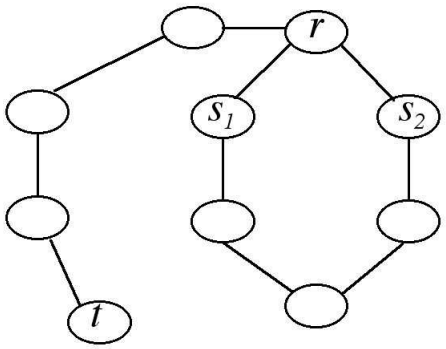


Figura 13.- Estructura "stem-and-cycle"

A partir de estas estructuras, mediante dos reglas diferentes, se irán obteniendo los siguientes eslabones de la cadena (ver figura 14). La primera regla consiste en añadir un arco que va desde la punta hasta un nodo dentro del ciclo y en eliminar alguno de los dos arcos que parten de ese nodo. La segunda consiste en añadir un arco que va desde la punta hasta un nodo del tallo y en eliminar el arco adecuado de los dos que parten de ese nodo.

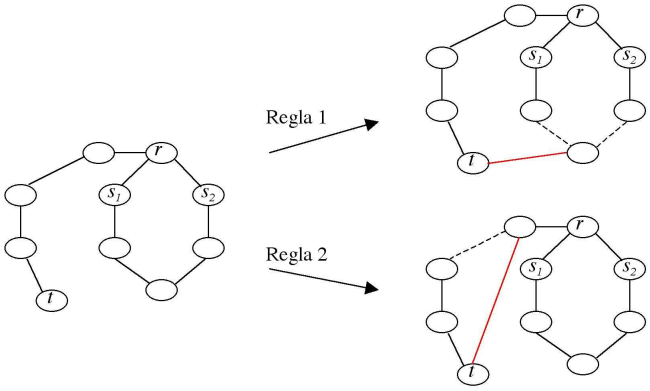


Figura 14.- Dos reglas para que obtener nuevas estructuras.

A medida que se vayan recorriendo eslabones en una cadena, en un determinado nivel puede ocurrir que la "stem-and-cycle" degenere, y en cualquier caso siempre se pueden obtener las llamadas soluciones prueba (que son ciclos factibles) de la Figura 15.-

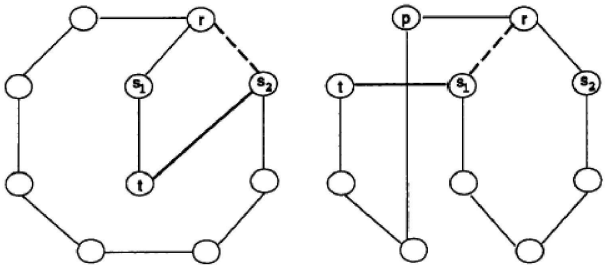


Figura 15.- Soluciones "prueba"

En resumen, se puede observar que las *Ejection Chains* dan lugar a unos vecindarios que incluyen los vecindarios de movimientos simples para a partir de ellos crear movimientos más complejos y poderosos. Los vecindarios definidos por *Ejection Chains* posibilitan movimientos de mayor poder sin un incremento significativo del esfuerzo computacional.

4.2 Estructura Double Routed (DR)

La podemos observar en la figura 16. Se parece a la anterior pero posee dos nodos "raíz". Glover (1996) muestra que esta estructura tiene varias ventajas sobre la anterior.

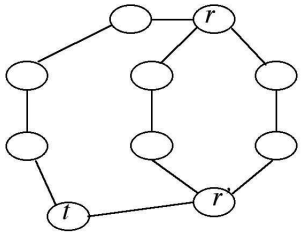


Figura 16.- Ejemplo de Double Routed

4.3 Estructura en flor

Es una generalización de la estructura SC, presentada también por Rego, (1998b). Esta estructura muestra su utilidad en el VRP sin ventanas de tiempo.

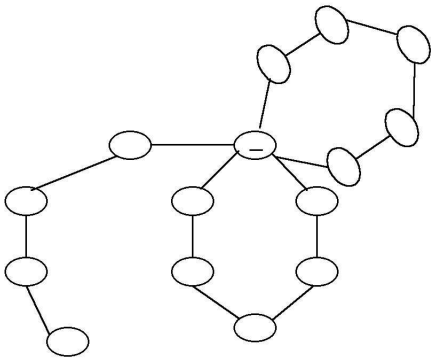


Figura 17.- Estructura en flor

4.4 Estructura Constrained Doubly Rooted, CDR

Sontrop et al. (2006) contribuyen con una nueva estructura de referencia que generaliza estructuras utilizadas previamente. La estructura de referencia que presentan llamada Constrained Doubly Rooted, CDR, es la que se muestra en la imagen.

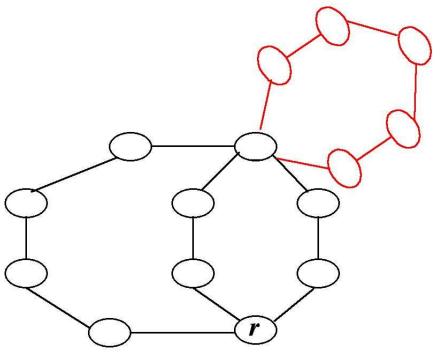


Figura 18.- Estructura Constrained Doubly Rooted

Tiene dos raíces. Se llama restringida porque el corazón, donde se sitúa el depósito siempre es una de las raíces.

5 Nuevas "Ejection Chains" para el Minmax VRP

En esta sección se describe el nuevo método tipo *"Ejection Chains"* que se propone en este trabajo. Para ello se describe primero la estructura de referencia y las diferentes reglas de transición en que se basan; es decir: para pasar de solución inicial a estructura, de estructura a estructura y de estructura a soluciones prueba. La estructura que se propone se describe en la figura 19. Es similar al a la estructura en flor descrita en la subsección 4.3, pero con arcos en vez de arista (es decir se trata de un grafo dirigido).

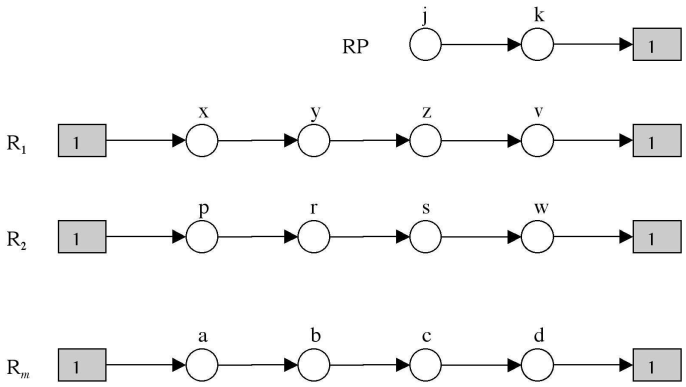


Figura 19.- Estructura de referencia propuesta, m rutas completas (R_j) y una ruta parcial incompleta o huérfana RP

Básicamente la estructura incluye todos los puntos del problema y consta de m rutas completas (en la figura 19 R_1, R_2 y R_m), es decir que empiezan y finalizan en el origen 1, y una ruta incompleta o parcial ("huérfana", RP), que también finaliza en el origen pero comienza en otro punto diferente.

Hay 2 formas de generar estas estructuras a partir de una solución. La primera, más sencilla, se ilustra en la figura 20: en una ruta se elimina un arco (en la figura el (y, z)) y se enlaza el primer punto de ese arco con el origen (se añade el arco $(y, 1)$)

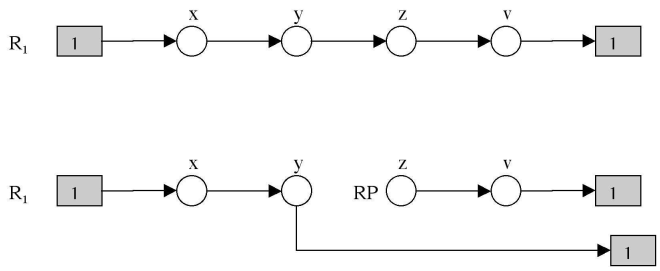


Figura 20.- Forma sencilla de crear una ruta parcial RP

En la segunda forma, se eliminan dos arcos en dos rutas diferentes, y se enlaza la primera parte de la primera ruta con la segunda parte de la segunda. La primera parte de la segunda ruta se enlaza con el origen 1. De esta manera como ruta parcial o "huérfana" la segunda parte de la primera ruta. La figura 21 ilustra este proceso.

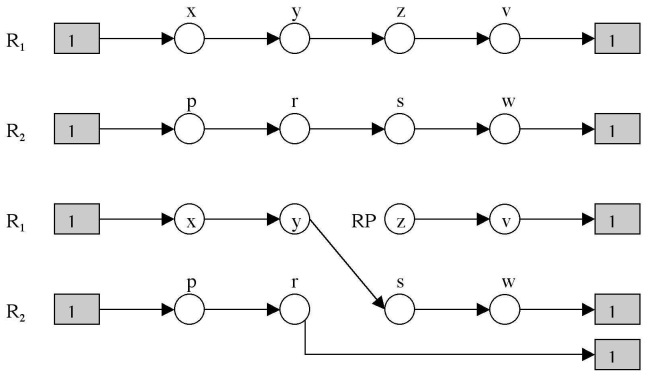


Figura 21.- Otra forma de crear una ruta parcial

La forma de pasar de una estructura a otra es la siguiente: se elimina un arco de una ruta y se enlaza la primera parte de esa ruta con la ruta parcial actual. De esta manera queda la segunda parte de la ruta como nueva ruta parcial. La figura 22 ilustra este proceso.

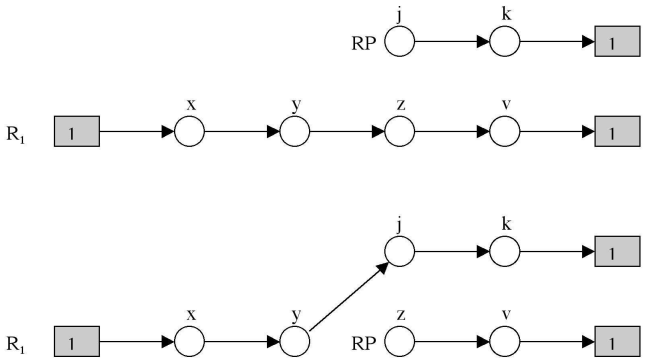


Figura 22.- Paso de una estructura a otra

Finalmente para generar soluciones prueba a partir de una estructura, se inserta la ruta parcial entre los dos últimos puntos de una ruta (i.e el penúltimo punto, y el origen). Por tanto se generan m soluciones prueba, tantas como rutas. La figura 23 ilustra este proceso.

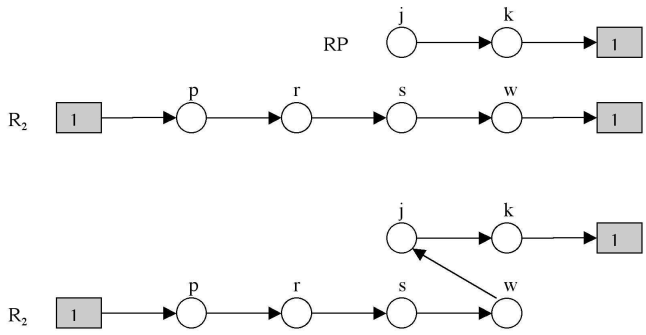


Figura 23.- Obtención de soluciones prueba

La transición de una estructura de referencia a otra debe evitar la creación de rutas parciales degeneradas, es decir, con solo un punto, (el destino final 1). La razón es que una estructura con ruta parcial degenerada da lugar a solo una solución prueba y además esta ya esta incluida entre las soluciones prueba obtenidas de la estructura anterior. Por otra parte para evitar ciclos, se ha de impedir las transiciones entre estructuras que supongan la incorporación de arcos eliminados en transiciones anteriores o en la creación de la primera estructura inicial.

Por tanto el procedimiento Ejection Chain trabaja de la forma siguiente:

- A partir de la solución inicial se crean todas las estructuras de referencia, ER, como se ha indicado anteriormente, y para cada una de ellas se generan las soluciones prueba ST correspondientes

- Entre todas ellas se identifica la mejor solución, según la función objetivo y su correspondiente estructura de referencia; se toma esta como primera estructura de referencia actual
- En los pasos siguientes se actúa de forma similar: desde la estructura de referencia actual se generan todas las estructuras de referencia posibles y sus correspondientes soluciones prueba; se toma como nueva estructura de referencia actual la correspondiente a la mejor solución
- El proceso finaliza cuando se han ejecutado un número predeterminado de pasos, aunque se pueden establecer otros criterios. La solución final obtenida (el output del proceso) es la mejor solución prueba visitada en todo el proceso.

Hay que hacer las siguientes consideraciones al proceso descrito anteriormente:

- En nuestro caso la función objetivo es la duración de la ruta más larga; en caso de "empate" se observaría cual es la segunda ruta más larga de cada solución; y si se mantuviera el empate la tercera y así sucesivamente
- La mejor solución se busca siempre entre las soluciones factibles, en otras palabras se prefiere siempre una solución factible que otra infactible. Si en algún paso no hubiera soluciones prueba factibles se elegiría como mejor solución aquella con menor grado de infactibilidad. Como en este problema las soluciones prueba solamente pueden violar las restricciones de capacidad, se elegiría como mejor solución aquella que menos exceda esta restricción en el conjunto de las rutas

6 Resultados Computacionales

A continuación, para contrastar la eficacia de estos movimientos tipo Ejection Chains, se van a realizar una serie de pruebas con las instancias reales de transporte escolar antes mencionadas. Para estas instancias se ejecutan sendos procedimientos de búsqueda local: uno que usa movimientos simples basados en CROSS intercambios (ver sección 3.3), y el segundo en el que el paso de una solución a otra se basa en el proceso de Ejection Chain descrito en el apartado anterior, con un número de pasos igual a 5. Los CROSS intercambios han sido usados con éxito para este problema y con estas instancias en Delgado y Pacheco (2001), Delgado (2002) y Pacheco y Martí (2006). En ambos procedimientos de Búsqueda Local tras cada movimiento las rutas de la nueva solución obtenida se mejoran con intercambios de Or (ver sección 2.2).

Las instancias reales se refieren a la recogida de alumnos de secundaria y su traslado a 16 institutos de secundaria en la provincia de Burgos; los datos para cada instituto (número de puntos de recogida, número de alumnos en cada punto matrices de distancia y tiempos) son descritas exhaustivamente en Delgado

(2002), además están disponibles en las páginas web de Rafael Martí y Joaquín Pacheco. Para cada instituto se han considerado diferentes números de autobuses m, variando desde $nvr - 1$, hasta $nvr + 3$, siendo nvr el número de vehículos usados en la realidad por las autoridades.

Las soluciones iniciales de las que parten ambos procedimientos de búsqueda local se generan con la adaptación del algoritmo de Fisher y Jaikumar (1981) propuesta en Pacheco y Martí, (2006). En la tabla 1 se muestran los resultados de ambos algoritmos de búsqueda local (BL_CROSS búsqueda local basada en CROSS intercambios y BL_EC basado en Ejection Chains), así como la función objetivo de la solución usada por las autoridades (Sol.Real).

Pr.		BL CROSS					BL EC				
	Sol.Real	M					m				
	$nvr/tmax$	$nvr-1$	nvr	$nvr+1$	$nvr+2$	$nvr+3$	$nvr-1$	nvr	$nvr+1$	$nvr+2$	$nvr+3$
$S1$	12/70	57	52	48	47	48	55	52	48	47	48
$S2$	5/45	47	44	**32**	32		47	42	33	32	
$S3$	6/60	54	45	43	39		54	45	43	39	
$S4$	3/70		57	41	38	36		**55**	41	38	**34**
$S5$	4/60	59	47	39			59	47	39		
$S6$	4/80	90	66	**54**	50	48	90	66	55	50	**43**
$S7$	6/60	54	45	37	36		**51**	45	37	36	
$S8$	9/75	61	58	50	49	45	**60**	58	50	**47**	**44**
$S9$	5/90	**91**	65	57	51	47	93	65	**55**	**50**	47
$S10$	6/60	48	44	40			**47**	**43**	40		
$S11$	4/60	67	51	45	39		**65**	**48**	**43**	39	
$S12$	2/25		15	14	9			15	14	9	
$S13$	6/45	40	36	29	29		40	36	29	29	
$S14$	5/60	53	46	38			53	46	37		
$S15$	7/50	50	45	44	40		50	**44**	**42**	40	
$S16$	2/60	84	51	40	35		84	51	40	35	

Tabla 1: Resultados obtenidos por ambos procedimientos de búsqueda local (en minutos)

De la tabla 1 que muestra los resultados computacionales se pueden extraer las siguientes conclusiones:

- Los resultados obtenidos por ambas estrategias mejoran los resultados que se usaron en la realidad. Esta mejora es significativa en casi todos los casos; solamente en $S2$ la diferencia no es muy grande (1 y 3 minutos).
- Los resultados obtenidos por ambas estrategias son iguales en la mayoría de

las instancias 43 de 64; en muchos de estos 43 casos la solución final coincide con la mejor solución reportada hasta la fecha

- En el resto de los 21 instancias las Ejection Chains obtienen mejor resultado en 18 casos y los CROSS intercambios en 3
- Además las Ejection Chains mejoran significativamente en algunos casos ($S6$ $m = 7$ en 5 minutos; $S11$ $m = 4$, en 3 unidades, y varios casos en 2 minutos); mientras que los CROSS intercambios simples mejoran en 1 minuto en 2 casos ($S2$ $m = 6$; y $S9$ $m = 4$) y en 2 minutos en 1 caso ($S6$ $m = 5$).

Por tanto es claro que aunque no muestren una superioridad aplastante las Ejection Chains propuestas en este trabajo dan ligera pero significativamente mejores resultados que los movimientos simples propuestos en anteriores trabajos para este problema.

7 Conclusiones

En este trabajo se diseñan movimientos basados en Ejection Chains para problemas de rutas, concretamente para el Minmax VRP. Nuestras Ejection Chains se basan en estructuras de referencia y reglas de transición relativamente sencillas, fáciles de programar y con una complejidad polinomial en el tamaño del problema ($\theta(n2)$): m rutas completas y una ruta parcial que en cada paso va reinsertándose en una ruta completa dando lugar a otra ruta parcial. Los movimientos vecinales a que dan lugar estas Ejection Chains mejoran significativamente los movimientos simples usados para este problema en las instancias reales de transporte escolar en referencias recientes. Hay que indicar que al menos en un contexto de búsqueda local. Es de esperar que este tipo de movimientos mejoren aún más sus resultados insertados en estrategias heurísticas más sofisticadas que una búsqueda local. Además las estructuras y reglas usadas podrían sofisticarse más para dar lugar a movimientos más potentes y adaptarse fácilmente a otros modelos de rutas.

8 Bibliografía

[1] Bräysy O. (2003). "A Reactive Variable Neighborhood Search for the Vehicle-Routing Problem with Time Windows". INFORMS Journal on Computing. Vol. 15, No. 4, pp. 347–368

[2] Cao B. and Glover F. (1997). "Tabu search ejection chains – application to a node weighted version of the cardinality- constrained TSP". Management Science 43, No.7, 908-921

[3] Clarke G. and Wright J. (1964) "Scheduling of vehicles from a central depot to a number of delivery points", Operations Research, 12 Vol.4, 568-581

[4] Corberán, A., Fernández E., Laguna M. and Martí R. (2002) "Heuristic Solutions to the Problem of Routing School Buses with Multiple Objectives" Journal of the Operational Research Society, vol. 53, nº 4, pp. 427 – 435

[5] Cordeau J.-F., Laporte G. and Mercier A. (2001) "A unified tabu search heuristic for vehicle routing problems with time windows" Journal of the Operational Research Society 52, 928-936

[6] Delgado C. (2002). Nuevas Técnicas Metaheurísticas: Aplicación al Transporte Escolar. Servicio de Publicaciones de la Universidad de Burgos. ISBN 84-95211-62-9

[7] Delgado C. and Pacheco J. (2001) "Minmax Vehicle Routing Problems: Application to School Transport in the Province of Burgos (Spain)", Lecture Notes in Economics and Mathematical Systems. Springer, Berlin, nº 505, pp. 297-318

[8] Fisher M. L. (1994) "Optimal Solution of Vehicle Routing Problems Using Minimum K-trees", Operations Research 42, 626-642

[9] Fisher M.L. and Jaikumar R. (1981) "A Generalized Assignment Heuristic for Vehicle Routing", Networks, vol.11, nº 2, 109-124.

[10] Gambardella L. M., Taillard E. and Agazzi G. (1999) "MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows", In D. Corne, M. Dorigo and F. Glover, editors, New Ideas in Optimization. McGraw-Hill.

[11] Gendreau M., Hertz A. and Laporte G. (1994) "A Tabu Search Heuristic for Vehicle Routing Problem", Management Science 40 (10), pp.1276- 1290

[12] Gillet B.E. and Miller L. R. (1974) "A Heuristic Algorithm for the Vehicle Dispatch Problem", Operations Research, 22, 340-349

[13] Glover F. (1992) "New Ejection Chain and Alternating Path Methods for Traveling Salesman Problems", Computer Science and Operations Research, pp. 449-509.

[14] Glover F. (1996) "Ejection chains, reference structures and alternating path methods for traveling salesman problems", Discrete Applied Mathematics, 65(1-3) pp. 223–253

[15] Kontoravdis G. and Bard J.F. (1995) "A GRASP for the Vehicle Routing Problem with Time Windows", ORSA Journal on Computing. Vol. 7, pp 10-23.

[16] Lin S. (1965) "Computer Solutions to the Traveling Salesman Problem", Bell Syst. Tech. Journal Vol. 44, pp 2245-2269.

[17] Lin S. and Kernighan B. W. (1973) "An Effective Heuristic Algorithm for the Traveling Salesman Problem", Operations Research. Vol. 20, pp. 498-516.

[18] Or, I. (1976) "Traveling Salesman Type Combinatorial Problems and their Relations to the Logistics of Blood Banking." Ph. Thesis. Dpt. of Industrial Engineering and Management Sciences, Northwestern Univ.

[19] Osman, I. H. (1993) "Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem", Annals of Operations Research. Vol. 41, pp 421-451

[20] Pacheco J. and Martí R. (2006). "Tabu Search for a Multi-Objective Routing Problem", Journal of Operations Research Society, 57, 29-37

[21] Pesch E. y Glover F. (1997) "TSP ejection chains", Discrete Applied Mathematics 76, No.1-3, 165-181

[22] Potvin J.Y. and Bengio S. (1994) "A Genetic Approach to the Vehicle Routing Problem with Time Windows", Technical Report CRT-953. Centre de Recherche sur les Transports. Univ. Montréal.

[23] Rego C. (1998a) "Relaxed tours and path ejections for the traveling salesman problem", European Journal of Operational Research, Volume 106, 2-3, 16, pp. 522-538.

[24] Rego C. (1998b) "A Subpath Ejection Method for the Vehicle Routing Problem", Management Science Vol 44, nº 10, pp. 1447-1459.

[25] Rego C. (2001). "Node-ejection chains for the vehicle routing problem: Sequential and parallel algorithms". Parallel Comput. 27, No.3, 201-222.

[26] Rego C. and Roucairol C. (1996). "A parallel tabu search algorithm using ejection chains for the vehicle routing problem". Osman, Ibrahim H. (ed.) et al., Meta-heuristics: theory and applications. International conference (MIC), Breckenridge, CO, USA, 22–26 July 1995. Dordrecht: Kluwer Academic Publishers. 661-675.

[27] Rochat Y. and Taillard . D. (1995) "Probabilistic Diversification and Intensification in Local Search for Vehicle Routing", Journal of Heuristics, 1, 147-167

[28] Sontrop, H. M. J. , van der Horn S. P. ,Teeuwen G. , and Uetz M. (2005) "Fast Ejection Chain Algorithms for Vehicle Routing with Time Windows" in Hybrid Metaheuristics, Lecture Notes in Computer Science. Volume 3636 Pages 78-89 Springer Berlin / Heidelberg

[29] Taillard E., Badeau P., Gendreu M., Guertain F. y Potvin J.Y. (1997) "A Tabu Search heuristic for the Vehicle Routing Problem with Time Windows", Transportation Science. Vol. 31, pp 170-186.

[30] Thangiah S. R., Osman I. H., and Sun T. (1994) "Hybrid Genetic Algorithm, Simulated Annealing, and Tabu Search methods for the Vehicle Routing Problem with Time Windows", Working paper UKC/OR94/4. Institute of Mathematics and Statistics. University of Kent, Canterbury.

[31] Toth P. and Vigo D. (2001) "Branch-and-bound algorithms for the capacitated VRP". In: Toth, P., Vigo, D. (Eds.), The Vehicle Routing Problem. SIAM: Philadelphia, pp. 29-52

[32] Van Breedam A. (1995) "Improvement heurisitcs for the vehicle routing problem based on simulated annealing", European Journal of Operational Research 86, 480-490

[33] Vondouris C. and Tsang E. (1999) "Guided Local Search for the Traveling Salesman Problem", European Journal of Operations Research. Vol. 113, pp 469-499.

Planificación de la producción en una empresa de contrachapado

Vicente Valls[a], Francisco Ballestín[b], Pilar Lino[c], ngeles Pérez[c], Sacramento Quintanilla[c]

[a] Vicente.Valls@uv.es, Departamento de Estadística e Investigación Operativa, Universitat de València

[b] Francisco.Ballestin@unavarra.es, Departamento de Estadística e Investigación Operativa, Universidad Pública de Navarra

[c] Pilar.Lino@uv.es, Angeles.Perez@uv.es, Maria.Quintanilla@uv.es, Departamento de Matemáticas para la Economía y la Empresa, Universitat de València

1 Introducción

En este trabajo se presenta un sistema de apoyo a la gestión y control de la producción de una pequeña empresa de contrachapado de la Comunidad Valenciana. La producción de la empresa se gestionaba de manera intuitiva, basándose únicamente en la experiencia de los directivos. El alto grado de incumplimiento de las fechas de entrega de los pedidos señalaba la ineficiencia de su gestión y la necesidad de instalar un sistema de gestión integral de la producción.

La producción en esta empresa es un proceso continuo con dos fases bien diferenciadas en cuanto a la incertidumbre de sus procesos. La primera de ellas es la que contiene mayor grado de incertidumbre, por la influencia de factores externos no controlables. La segunda es más determinista y por ello su planificación es susceptible de ser automatizada. Al final de esta segunda fase, se detectó un cuello de botella que limitaba el ritmo de trabajo previo.

A partir de un análisis exhaustivo de la empresa se decidieron dos tipos de

actuaciones para mejorar la eficiencia del sistema productivo. Primero, desarrollar un procedimiento matemático para resolver de forma eficiente el problema originado por el cuello de botella. En segundo lugar, diseñar una aplicación que integrase todos los datos del sistema productivo (clientes, proveedores, stocks, pedidos, subproductos generados, horas utilizadas, desperdicio de materia prima) junto con el algoritmo matemático. Finalmente, debía proveerse al gestor de diferentes informes que le permitiesen planificar de manera más eficiente la parte del sistema productivo no susceptible de ser automatizada. Estos informes proporcionan información sobre el estado de los pedidos, rendimiento de las distintas secciones, estados de stocks, calidad de la materia prima por proveedor, etc.

En los siguientes epígrafes se describe la empresa y su proceso productivo; el sistema de información desarrollado (GESPLAN) y sus componentes principales: el algoritmo heurístico para la automatización de la planificación y la base de datos que gestiona toda la información. Finalmente se exponen las conclusiones.

2 Descripción del proceso productivo

La empresa se dedica a la elaboración de tablero contrachapado, destinado mayoritariamente a la obtención de los componentes utilizados por los fabricantes de envases de madera. También produce tablero para la fabricación de mueble escolar y de esquís. Su principal actividad consiste, pues, en la transformación de troncos de chopo en tableros de contrachapado, que posteriormente se cortan a la medida solicitada por el cliente. La producción se realiza contra pedido y también para stock.

En el proceso de transformación (Figura 1), la madera pasa secuencialmente por 4 secciones: desenrollo, secado, prensado y corte. En la primera sección, los troncos de chopo se desenrollan y cortan en hojas de chapa cuadradas o rectangulares de diferentes medidas y grosores. El aprovechamiento depende de diversos factores como la forma del tronco, el grado de humedad, el grosor de la corteza,... La eliminación de la humedad de la madera, en la sección de secado, se realiza bien en una máquina o bien en el secadero exterior al aire libre. Las hojas secas se clasifican en tres grupos según su calidad (primera, segunda y tercera). A continuación, estas hojas de chapa clasificadas pasan a la sección de prensado, en donde se agrupan y encolan varias hojas y se introducen en las prensas obteniéndose el tablero contrachapado. Los tableros de idénticas características se apilan formando los *lotes*, que se etiquetan con la medida, cantidad, fecha y prensa utilizada. En la sección de corte, los tableros son lijados y cortados para obtener el producto final solicitado por el cliente. Cada tablero pasa secuencialmente por dos máquinas. La primera (*macro*) procesa los tableros uno a uno y realiza simultáneamente varios cortes longitudinales para obtener piezas con el ancho indicado en el pedido. Las piezas así obtenidas se apilan y se cortan en la *retestadora* a la medida del largo solicitado para el producto final. Esta máquina

puede realizar también varios cortes simultáneos. La empresa dispone de dos macros y una retestadora. En todas ellas, las sierras laterales cortan una medida fija de cada lado que no se aprovecha.

La producción es por tanto un proceso lineal y continuo en el que se diferencian dos fases en función de la incertidumbre de los datos que intervienen en el proceso. La primera fase (secciones de desenrollo y secado) es la que mayor grado de incertidumbre posee. En estas secciones no se conocen con precisión la duración del proceso ni la calidad del producto obtenido (hojas de primera, segunda o tercera calidad), ya que son numerosos los factores externos que influyen en él: calidad de la madera recibida, situación climatológica, pues de ésta dependen los tiempos de secado exterior,... La segunda fase (secciones de prensado y corte) puede ser planificada de forma automática pues los datos son deterministas y conocidos. El objetivo de la planificación será satisfacer las fechas de entrega de los pedidos y optimizar la utilización de los recursos productivos. Una vez planificada esta segunda fase, el gestor puede planificar de manera más eficiente la parte del sistema productivo no susceptible de ser automatizada.

El estudio de equilibrado del flujo de la producción ha permitido identificar el cuello de botella en la sección de corte. Concretamente, las principales limitaciones se producen en la máquina que realiza los últimos cortes en el tablero (retestadora). El tiempo de preparación de esta máquina para ajustar las sierras cuando se cambia la medida del producto es muy elevado, pues los cambios y mediciones se realizan manualmente. Por otra parte, la capacidad productiva de las prensas es suficiente para alimentar la sección de corte.

Figura 1: Esquema de producción de la empresa de contrachapado

3 Sistema de información para la gestión integral GESPLAN

Con el objetivo de mejorar la gestión integral de la empresa, se ha desarrollado el sistema de información, GESPLAN, que a continuación se describe. En él se integra una base de datos con los procedimientos automatizados de planificación, que se alimentan mutuamente. La información global resultante permite al gestor planificar de manera más eficiente la parte del sistema productivo no susceptible de ser automatizada. Además, facilita la elaboración de informes para los encargados de las distintas secciones y de los documentos administrativos habitualmente necesarios.

El objetivo principal del sistema de planificación es satisfacer las fechas de entrega de los pedidos de los clientes, optimizando la utilización de los elementos productivos. Para ello, dado que la retestadora es un cuello de botella, el sistema comienza elaborando un plan de corte para dicha máquina. Con la información residente en la base de datos relativa a los pedidos pendientes y al stock almacenado, se ejecuta el algoritmo heurístico que se describe en el epígrafe siguiente. Como resultado, se obtiene las dimensiones y el número de los tableros a cortar, el patrón de corte de cada tablero (número de sierras y distancia entre ellas en cada máquina de corte) y el orden en que deben pasar los lotes de tableros por la sección de corte. También se indica lo más tarde que se debe cortar cada lote para que se puedan cumplir las fechas de entrega. Además del objetivo general del sistema, el algoritmo también persigue minimizar desperdicios y minimizar el tiempo total de proceso.

Con el fin de poder llevar a cabo el plan de corte generado por el algoritmo, debe organizarse el trabajo de las tres secciones previas para que los lotes de tableros que salen de las prensas alimenten de manera adecuada la sección de corte. Para ello, a partir de los resultados del algoritmo, GESPLAN construye también un plan para cada una de las secciones anteriores.

El plan de prensado indica las dimensiones y cantidad de tableros de los lotes y el orden en que han de ser prensados, así como lo más tarde que debe comenzar el proceso de cada lote para satisfacer las fechas de entrega. El plan de troceado de troncos y de secado indica el número de hojas necesarias de cada calidad, las dimensiones de las hojas y el orden de corte y secado. No se puede indicar con precisión cuántas hojas hay que cortar ni cuántos troncos hay que trocear, aunque sí se puede hacer una estimación.

Toda la información generada se vierte en la base de datos, facilitando así el posterior análisis por parte la empresa. Esta información se extrae mediante informes ya programados en la base de datos. Estos informes están parametrizados, permitiendo variar fechas, proveedores, clientes, etc. El beneficio es doble, por una parte GESPLAN permite elaborar automáticamente albaranes, facturas, información sobre pedidos no completos, etc. y por otra planificar automáticamente

la producción de la segunda fase del proceso y controlar la producción no susceptible de ser planificada de manera automática. La comunicación entre los dos componentes principales del sistema, base de datos y algoritmo, es por tanto bidireccional. La base de datos ha sido creada con el programa Microsoft Access© con procedimientos para exportar e importar datos a la segunda componente del software. El algoritmo diseñado para planificar la producción es un algoritmo heurístico específicamente desarrollado para esta aplicación.

4 Algoritmo heurístico para la planificación de la máquina de corte

El algoritmo heurístico desarrollado calcula las dimensiones de las hojas que hay que desenrollar e intenta encontrar una secuencia de paso por la retestadora que permita satisfacer las fechas de entrega pactadas con los clientes.

El algoritmo resuelve el problema de planificar la máquina de corte en tres etapas. En la primera (apartado 4.1) decide los lotes de tableros que se tienen que procesar para realizar todos los pedidos. En segundo lugar (apartado 4.2) optimiza el orden en que dichos lotes pasan por la retestadora. Por último (apartado 4.3), la solución calculada se propaga al resto de máquinas.

4.1 Elección de los lotes para atender a los pedidos

Definimos un pedido de manera que todas las piezas sean exactamente del mismo tipo, con las mismas características. En general, será necesario dividir las peticiones reales de un cliente en diversos pedidos para que se cumpla esta premisa. En esta primera etapa del algoritmo se determina el número de tableros que debe ser procesado y las medidas de los mismos para obtener las piezas demandadas en los pedidos, intentando minimizar el residuo generado (parte del tablero que se desperdicia). De cada tablero van a surgir piezas de cómo máximo dos pedidos, A (principal) y B (residual). Definimos un lote como un conjunto de tableros que serán cortados de la misma manera, dando lugar cada tablero a n_1 piezas de las mismas medidas para el pedido A y, si procede, n_2 piezas de otras medidas para el pedido A o para stock. Por lo tanto, cada lote queda determinado por el número de tableros, las dimensiones de los mismos, el tipo de acabado, el número de sierras en las distintas máquinas de corte y la distancia entre ellas. Además, a cada lote se le asigna la fecha de entrega del pedido principal asociado.

La empresa trabaja con una serie de medidas estándar de tablero. Los pedidos se ordenan por fecha de entrega creciente y para cada pedido A se busca aquella longitud de tablero que se pueda convertir en piezas del pedido de manera que el desperdicio sea mínimo. Después, se calcula la parte sobrante de tablero, descontados los márgenes, para ver si de ella se puede obtener alguna pieza de otro

pedido B que requiera el mismo acabado o, en su defecto, alguna pieza para stock. Obviamente esto reduce significativamente el número de pedidos que pueden utilizarse como pedidos residuales. Por último, se calcula el número de tableros necesario de la longitud escogida de manera que sirvamos por completo al pedido A o al B. Si con los tableros que componen el lote no servimos completamente al pedido A, calcularemos otro lote para las piezas restantes.

Por lo tanto, cada lote tiene asociados como máximo dos pedidos y cada pedido puede requerir más de un lote.

4.2 Optimización del orden de los lotes

Una vez decididos los lotes que vamos a procesar, debemos escoger el orden en que van a ser atendidos por la retestadora. Para evaluar la bondad de un orden concreto tenemos en cuenta dos factores. El primero y más importante es si los pedidos van a poder ser entregados a tiempo. Para calcular esto sólo tenemos en cuenta el tiempo empleado por la retestadora, ya que según la experiencia ese tiempo va a ser mayor que el utilizado en el resto de secciones de la empresa. Cada pedido entregado fuera de tiempo lleva asociada una penalización dependiente de su importancia y de la tardanza en entregarlo. El segundo factor que tenemos en cuenta, con mucho menos peso que el primero, es el tiempo necesario para los cambios de sierra en la retestadora. El problema es por tanto encontrar el orden de los lotes que minimice esta función de penalización. Veamos el modelo que corresponde con este problema de optimización. Para simplificar la formulación supondremos que cada lote sirve a un único pedido, aunque sí permitiremos varios lotes por pedido.

Modelo Partimos de la definición de los n lotes, $l_1, \ldots, l_n$. Cada pedido $P_h, h = 1, \ldots, p$, lleva asociados n_h lotes necesarios para terminar el pedido, $P_h = \{l_{P_h(1)}, \ldots, l_{P_h(n_h)}\}$.

Definimos $x_i, i = 1, \ldots, n$, como el instante en que el lote i-ésimo comienza a ser tratado por la retestadora. Entonces $x_i + d_i$ es el instante de finalización del lote, donde d_i denota el tiempo necesario para cortar todos los tableros del lote con la retestadora, una vez sus sierras están en la posición adecuada. Esta duración del lote es conocida y constante (una vez definidos los lotes). La fecha de finalización del pedido P_h es por tanto $f_h = max\{x_{P_h(i)} + d_{P_h(i)}, i = 1, \ldots, n_h\}$. Nuestro principal objetivo es entregar los pedidos antes de su fecha de entrega dd_h. Una tardanza de un pedido conlleva una penalización c_h dependiente de la importancia del mismo. El primer y más importante factor de la función objetivo es por tanto

$$\sum_{h=1}^{p} c_h (f_h - dd_h)^+$$

Según la información proveniente de la empresa existe una gran diferencia en los tiempos necesarios para el cambio de sierras entre dos lotes, dependiendo del tamaño de los tableros de ambos lotes y las sierras que se deban emplear. Es por tanto importante intentar minimizar el tiempo total de cambio de sierras. Dado que este tiempo depende del orden concreto de lotes que escojamos, estamos trabajando con un recurso con tiempo de preparación dependiente de la secuencia. Los tiempos de preparación son típicos en los procesos industriales, donde las unidades de proceso como reactores o filtros tienen que ser limpiados después de la finalización de ciertas actividades. En general, el tiempo de limpieza será mayor para pasar de un producto de baja calidad a uno de alta que viceversa. En nuestro caso el tiempo de preparación depende de si es necesario mover, añadir o quitar sierras. Para profundizar en la literatura sobre tiempos de preparación, consultar [1]o [8, Sec. 3]. La definición de lotes nos permite calcular la matriz **cambio**(), cuya elemento ij contiene el tiempo de preparación de la retestadora entre los lotes i y j.

Para modelizar las restricciones del modelo necesitamos conocer el orden en que los lotes son procesados. Concretamente, definimos para cada par de lotes $(i,j), i \neq j$, la variable binaria y_{ij}, que es 1 si el lote i se procesa inmediatamente antes que el j, 0 en caso contrario. Si un lote j se procesa justo después del i la restricción correspondiente es $x_j = x_i + d_i + cambio(i,j)$, dado que empezaremos a procesar el lote j cuando se hayan efectuado los cambios de sierra correspondientes después de acabar el lote i. Esta igualdad se consigue añadiendo 2 restricciones por cada pareja de lotes i y j, siendo M una constante muy grande, por ejemplo $M = \sum_{i=1}^{n} d_i + \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} cambio(i,j)$:

$$x_j \geq x_i + d_i + cambio(i,j)y_{ij} - M(1 - y_{ij}) \qquad (1)$$

$$x_j \leq x_i + d_i + cambio(i,j)y_{ij} + M(1 - y_{ij}) \qquad (2)$$

Un valor de 1 en y_{ij} nos lleva a la igualdad deseada, mientras que un valor de 0 nos ofrece dos desigualdades triviales. Estas variables nos ayudan además a modelizar la segunda parte de la función objetivo, minimizar el tiempo invertido en cambio de sierras, $\sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} cambio(i,j)y_{ij}$. Como éste juega un papel secundario, lo multiplicamos por una constante β mucho menor que α, la constante por la que multiplicamos el primer factor. El modelo de programación lineal entera queda resumido en la Figura 2.

$$
\begin{array}{lll}
Min & \alpha \sum_{h=1}^{p} c_h (f_h - dd_h)^+ + \beta \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} cambio(i,j) y_{ij} & \\
s.a. & x_j \geq x_i + d_i + cambio(i,j) y_{ij} - M(1 - y_{ij}) & i,j = 1,\ldots,n,\ i \neq j \\
 & x_j \leq x_i + d_i + cambio(i,j) y_{ij} + M(1 - y_{ij}) & i,j = 1,\ldots,n,\ i \neq j \\
 & f_h \geq x_{P_h(i)} + d_{P_h(i)} & i = 1,\ldots,n_h \\
 & & h = 1,\ldots,p \\
 & x_i \geq 0 & i = 1,\ldots,n \\
 & y_{ij} \in \{0,1\} & i,j = 1,\ldots,n,\ i \neq j \\
 & \sum_{j=1, j\neq i}^{n} y_{ij} = 1 & i = 1,\ldots,n
\end{array}
$$

Figura 2. Modelo matemático

Este problema es NP-duro, al reducirse al problema del agente viajero (TSP) cuando $\alpha = 0$ y $\beta = 1$. Cuando $\alpha = 1$ y $\beta = 0$ y $c_h = 1$, obtenemos el problema de minimización de la suma de tardanzas en una máquina, también NP-duro [3]. En ambos caso hemos supuesto que tenemos un único lote por pedido.

Se ha diseñado un algoritmo heurístico para obtener una solución de este problema. Sin embargo, no vamos a intentar encontrar una solución óptima global para el mismo. Nuestras estimaciones sobre el tiempo empleado para cada acción son muy aproximadas. Además, el índice de percances que puede ocurrir es alto. Por tanto, no podemos controlar con completa seguridad si un pedido lo podemos entregar a tiempo o no, i.e., la función objetivo (primer factor) no es completamente fiable, tiene demasiada incertidumbre. Para paliar esta carencia vamos a restringir el espacio de búsqueda de soluciones, limitándonos a aquellas donde los pedidos sigan un orden aproximado acorde con la fecha de entrega, permitiéndonos variaciones para tener en cuenta la importancia de los pedidos y poder mejorar la segunda parte de la función objetivo entre aquellos subórdenes que conduzcan a entregar los pedidos a tiempo. Esquema básico del algoritmo.

La Figura 3 muestra el esquema algorítmico básico compuesto por dos meta-heurísticos, el primero de ellos es un GRASP y el segundo un algoritmo genético. Los algoritmos genéticos (GAs) fueron introducidos en [7] y simulan la evolución biológica. Trabajan con conjuntos de soluciones denominados poblaciones que van cambiando o evolucionando a lo largo del tiempo. La calidad de una solución le ayuda a sobrevivir las distintas generaciones o ser reemplazada por nuevas soluciones mejores (supervivencia del más fuerte), calculadas a partir de viejos individuos mediante combinación y mutación. Para una introducción en los GAs consultar [5].

1. Definir lotes. Sea n el número de lotes.
2. POB = GRASP($npob$).
3. SOLUCIN = GA(POB, $npob$, $nIter$).
4. Propagar SOLUCIN al resto de secciones.

Figura 3. Esquema algorítmico GRASP + GA($npob$, $nIter$)

El algoritmo desarrollado comienza calculando una población inicial POB que contiene $npob$ individuos. A partir de entonces se repite el mismo proceso un número prefijado de veces $nIter$ (ver Figura 4). La población se divide aleatoriamente en parejas de miembros. Cada pareja de individuos (padres) se combina o cruza, dando lugar a dos nuevas soluciones (hijos). Posteriormente, los hijos son sometidos a un proceso de mutación, tras lo cual se determina su calidad mediante la función objetivo. Al igual que en algunos algoritmos genéticos híbridos (ver [9]), y al contrario que en los genéticos puros, aplicamos una búsqueda local, DOSOPT, a cada uno de los hijos para mejorar su calidad (ver Figura 4). Por último, los añadimos a la población, obteniendo una población aumentada de $2npob$ individuos una vez hemos combinado todas las parejas formadas. Al eliminar los $npob$ peores individuos retornamos al tamaño original y podemos repetir el procedimiento.

1. Desde $i = 1, \ldots, nIter$:
   - 1.1 Dividir la población en pares de individuos
   - 1.2 Combinar cada par de individuos y generar otros dos individuos con el operador de cruce

     Para cada individuo λ obtenido en 1.2, hacer:

     λ = MUTACIN(λ)

     $\lambda = DOSOPT(\lambda)$

     $POB = POB \cup \{\lambda\}$

     Eliminar de POB los peores $nPob$ individuos, desempatando aleatoriamente
2. Devolver la mejor solución

Figura 4. Esquema algorítmico GA(POB, $npob$, $nIter$)

Individuos, calidad, cruce y mutación Un individuo I viene dado por una permutación de los lotes $I = (l_1^I, \ldots, l_n^I)$, que determina el orden en que los lotes

pasarán por la retestadora. La calidad de un individuo se calcula atendiendo a la función objetivo descrita en el modelo (Figura 2) y siguiendo este orden, donde el inicio del lote l_i^I es el final del l_{i-1}^I más $cambio(l_{i-1}^I, l_i^I)$. El final del lote l_i^I es el inicio más su duración, que al ser constante sólo se calcula una vez al inicio del algoritmo. El segundo factor de la función objetivo nos ayuda a desempatar entre (sub)permutaciones de lotes que lleven a las mismas tardanzas.

El cruce utilizado en el GA propuesto es el denominado de dos puntos para permutaciones [6] que a partir de dos individuos (madre y padre) genera dos nuevos individuos (hijo e hija) del siguiente modo: se escogen 2 enteros q_1 y q_2 con $1 \leq q_1 < q_2 \leq n$. La hija Ha se determina tomando los lotes de las posiciones $1, \ldots, q_1$ de la madre M, $l_i^{Ha} := l_i^M$. Las posiciones $i = q_1 + 1, \ldots, q_2$ se obtienen del padre P, $l_i^{Ha} := l_k^P$, donde k es el mínimo índice tal que $l_k^P \notin \{l_1^{Ha}, \ldots, l_{i-1}^{Ha}\}$. Las posiciones restantes $i = q_2 + 1, \ldots, n$ se obtienen de nuevo de la madre, $l_i^{Ha} := l_k^M$, donde k es el mínimo índice tal que $l_k^M \notin \{l_1^{Ha}, \ldots, l_{i-1}^{Ha}\}$. El hijo se obtiene de forma simétrica intercambiando los roles de la madre y el padre. Este operador de cruce ha resultado efectivo en varios problemas de secuenciación (ver por ejemplo [6] o [2]). La motivación para seleccionar este operador ha sido doble. En primer lugar, si ambas soluciones contienen un determinado lote en sus primeras posiciones, esta característica la heredarán ambos hijos. Esto es útil para aquellos lotes con fecha de entrega temprana, los cuales estarán en general en las posiciones delanteras en las buenas soluciones. En segundo lugar, la hija hereda con este operador muchos de los emparejamientos de la madre. Es decir, si dos lotes están en posiciones consecutivas en la madre (al menos en la primera parte, la anterior a q_1), también lo estarán en la hija. Esto es importante para el segundo miembro de la función objetivo, porque se conserva el tiempo de preparación de la retestadora.

La mutación es importante para introducir diversidad en la población. Se ha escogido una mutación similar a la empleada en [6]. Para cada una de las posiciones del individuo, salvo la última, se decide aleatoriamente si intercambiar el lote en esa posición con el situado en la siguiente. Debido a la incertidumbre en la función objetivo, no se permite que el pedido principal asociado al lote que se adelante tenga una fecha de entrega que supere en una cantidad δ (cuyo valor es elegido por el usuario) a la del pedido principal asociado al lote que se retrase. Población inicial y búsqueda local.

La población inicial se crea mediante la técnica GRASP (ver Figura 5). Un procedimiento de búsqueda miope aleatorizado y adaptativo (GRASP, [4]) es un proceso iterativo o multi-arranque, en el que cada iteración GRASP consiste en dos fases y calcula una solución. En general GRASP termina ofreciendo la mejor solución obtenida, aunque en el algoritmo que se presenta se han introducido todas las soluciones en la población inicial. En la fase constructiva de un GRASP se construye una solución posible iterativamente, añadiendo elemento a elemento. Los elementos se escogen atendiendo a una función greedy (miope o agresiva),

pero teniendo en cuenta la aleatoriedad. En la segunda fase o fase de mejora, se aplica una búsqueda local en un vecindario de la solución creada anteriormente.

1. Desde $i = 1, \ldots, nPob$
   - 1.1 $ELEG = 1, \ldots, n$
   - 1.2 Desde $i = 1, \ldots, n$
     - 1.2.1 $Fecha\_ent = min\{fecha\ de\ entrega(j),\ j \in ELEG\}$
     - 1.2.2 $ELEG2 = \{j \in ELEG / \mid fecha\ de\ entrega(j) - Fecha\_ent \mid < \mu\}$
     - 1.2.3 Calcular $tiempo(j),\ j \in ELEG2$; donde $tiempo(j)$ = tiempo de preparación (cambios de sierra) de la máquina si escogiéramos ahora el lote j
     - 1.2.4 Seleccionar lote $k \in ELEG2$ con muestreo aleatorio sesgado utilizando la función tiempo para calcular las probabilidades
     - 1.2.5 $ELEG = ELEG \setminus \{k\};\quad \lambda(i) = k$
   - 1.3 Hacer $\lambda = DOSOPT(\lambda)$
   - 1.4 $POB = POB \cup \{\lambda\}$
2. Devolver POB

Figura 5. Esquema algorítmico GRASP($nPob$) para calcular la población inicial

En la iteración i-ésima de la primera fase se escoge el lote que asignamos a la posición i-ésima del individuo. Para ello se restringen los lotes candidatos a ser seleccionados como aquellos lotes cuya fecha de entrega dista de la mínima fecha de entrega de los lotes elegibles una cantidad menor que un parámetro μ. Esta es la forma de asegurar que los lotes asociados a pedidos con una fecha de entrega temprana se sitúen en las primeras posiciones. A cada uno de los lotes candidatos a ser escogidos se le asigna una prioridad dependiente del tiempo de preparación necesario (cambios de sierra en la retestadora), teniendo en cuenta el lote escogido en la iteración anterior. Con estas prioridades se calculan probabilidades, que se utilizan para escoger el lote que ocupará la posición i-ésima del individuo que se está generando. De esta forma se cumplen dos características del GRASP, que la función que guía la selección es adaptativa y aleatorizada. El tiempo de preparación de la máquina sería la función greedy empleada en este caso.

En la segunda fase se aplica una búsqueda local, DOSOPT, basada en los dos-intercambios. En concreto, para cada posición $i, i = 1, \ldots, n-1$, estudiamos el impacto en la función objetivo producido si intercambiamos el lote l_i^I con el siguiente. Este cálculo es más sencillo a medida que aumenta i, dado que las fechas de finalización de los lotes anteriores se mantienen invariables. nicamente no estudiamos el intercambio de 2 lotes consecutivos si el pedido principal asociado

al lote l^I_{i+1} tiene una fecha de entrega que supere en una cantidad δ la del asociado al lote l^I_i. La razón es de nuevo la incertidumbre en la función objetivo. Esta búsqueda local se aplica también después en el genético a cada hijo calculado.

4.3 Propagación de la solución

Una vez decididos los lotes y el orden en que estos van a ser procesados aún es necesario realizar una serie de decisiones para propagar la solución al resto de secciones. Para ello ejecutamos un paso hacia atrás con cada uno de los lotes. En primer lugar, a cada lote le asociamos una de las dos macros existentes. En general escogeremos siempre la misma para dejar la otra libre, salvo que las características de los tableros requieran emplear la segunda. Calculamos cuántos tableros puede procesar la macro escogida al trabajar simultáneamente junto a la retestedora, puesto que esto es lo que generalmente se produce en la empresa. Como la retestadora es más rápida que la macro, a veces se necesitará cortar algunos tableros con la macro antes de que la retestadora comience con ese lote (son los que denominamos tableros rebalsados). Éstos se cortarán con la macro que generalmente está libre. Como disponemos de dos macros para abastecer a la retestadora, en principio la asignación se produce sin problemas.

El segundo y último paso de propagación consiste en la sección de prensado. Podemos procesar un lote con una, dos o las tres prensas de las que dispone la empresa, dependiendo de las características del lote. Empleamos un algoritmo totalmente greedy para decidir qué prensas utilizar. Para cada nuevo lote sabemos cuándo se libera cada prensa. Utilizando esta información estudiamos cuándo se terminaría de procesar cada lote y cuándo se liberaría cada prensa en cada una de las posibilidades (empleando una sola prensa, siendo la 1^a, 2^a ó 3^a, empleando la 1^a y la 2^a, ...). Escogemos aquella combinación con la que el lote salga antes de las prensas. Desempatamos por el número de prensas, escogiendo primero por tanto aquellas que antes dejan libre alguna(s) prensa(s).

5 Base de datos

La base de datos desarrollada en Microsoft Access© es una aplicación concebida para facilitar, mediante un entorno amigable y sencillo de utilizar, tareas como la actualización de información por el usuario, llamadas al programa de planificación y generación de informes. Es una base totalmente relacionada para evitar duplicidades e incoherencias en la información contenida.

El objetivo de la base de datos es doble. En primer lugar, debe de contener toda la información que requiere el algoritmo para poder obtener una solución. En segundo lugar, y dado que contiene todos los datos manejados por la empresa, debe generar automáticamente los informes que habitualmente realiza la empresa y que proporcionan información sobre facturación, albaranes, rendimiento de la

madera, etc. Describimos a continuación, presentando alguna de las pantallas, el contenido de la base de datos.

Desde el menú inicial, mostrado en la figura 6, se permite el acceso a todos los formularios diseñados para actualizar los datos que varían continuamente, a dos nuevos menús donde se encuentran respectivamente accesos a la creación de informes y formularios para la actualización de los datos permanentes; y por último, al programa para obtener la solución a partir de los datos referentes a los pedidos y al stock.

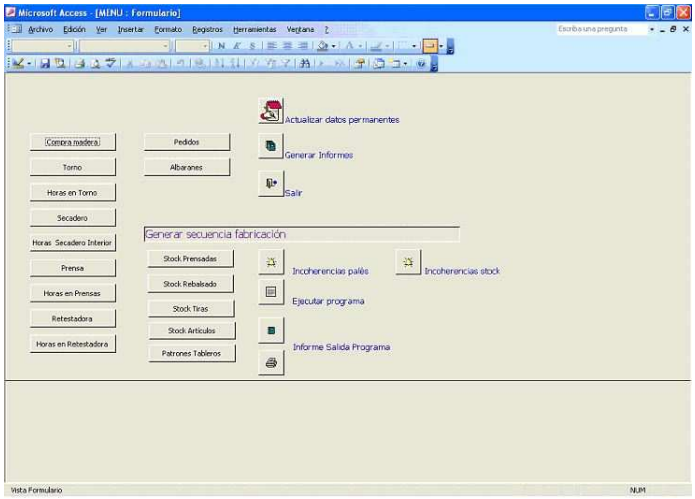


Figura 6 Menú inicial de GESPLAN

La primera columna de la pantalla mostrada en la figura 6 contiene el acceso a los formularios que se han desarrollado para actualizar toda la información del proceso productivo, tanto del material comprado (madera) como de los productos semielaborados que se obtienen a lo largo del proceso. Además, se accede también a sendos formularios en los que se imputan horas de trabajo en cada sección para elaborar posteriormente informes que faciliten el análisis del rendimiento del trabajo en cada una de ellas.

A la derecha de este menú, se observan dos partes separadas por una línea horizontal. En la parte superior, los accesos a pedidos y albaranes enlazan con respectivos formularios. En el primero de ellos (pedidos) se completa toda la información relativa a los pedidos que se reciben, desde el cliente que lo solicita hasta el detalle pormenorizado de la mercancía requerida. Los datos contenidos en el formulario albaranes están relacionados con los pedidos origen del albarán (un pedido puede desglosarse en varios albaranes), con los datos de stock, ya que la mercancía puede provenir de stock y con los artículos fabricados para dicho pedido. Parte de la información contenida en la tabla definida para albaranes

proporcionará los datos para confeccionar informes de dichos albaranes que serán enviados junto con el material requerido a los clientes.

Además del enlace para abandonar el sistema, aparecen otros dos (actualizar datos permanentes y generar informes) que conducen a sendos menús. El primero de ellos accede a los formularios desde los que se actualizan los datos que poco varían a lo largo del tiempo (datos de clientes, proveedores, tipos de productos e información relativa a los tipos de corte y máquinas utilizadas en el proceso productivo,...). El segundo contiene una serie de informes, diseñados para proporcionar información de gran utilidad a los gestores (pedidos pendientes de fabricar, pedidos fabricados no servidos, rendimiento de las secciones por periodo, facturación por cliente, facturación por periodo, stock,...).

La parte inferior del menú mostrado en la figura 6, está destinada a ejecutar con éxito el algoritmo desarrollado para planificar la sección de corte. Aparecen accesos a formularios que contienen los datos del stock, que deben de estar actualizados para obtener una solución correcta. Asimismo, se ofrece la posibilidad de acceder a informes en los que se muestra, si las hay, incoherencias encontradas en los datos. La ejecución del algoritmo determinará el plan de corte propuesto.

A continuación se muestran como ejemplo el formulario de los pedidos (figura 7), un informe que notifica los pedidos o partes de pedidos ya han sido fabricados y están pendientes de servir (figura 8) y otro que advierte sobre la rentabilidad que se ha obtenido de la madera adquirida, partida por partida (figura 9).

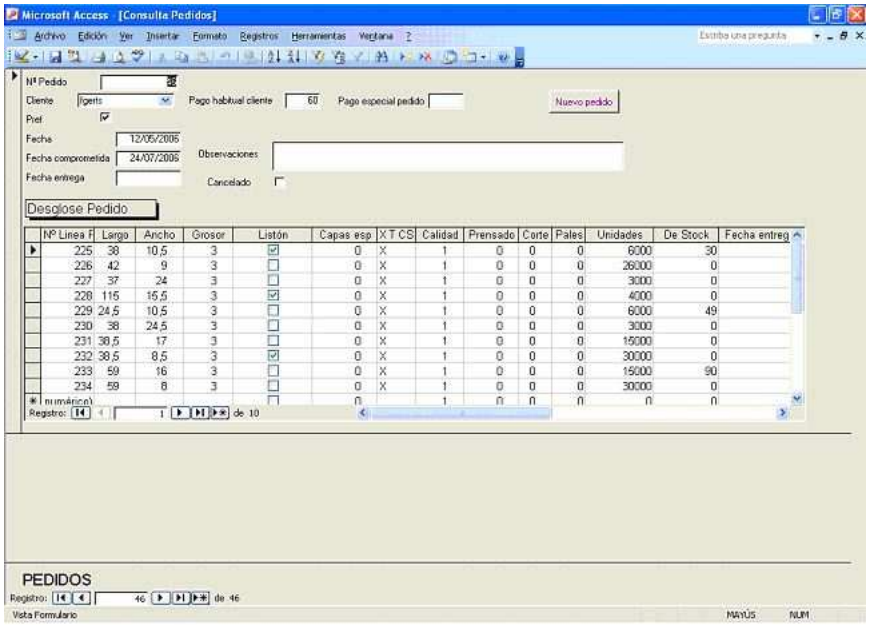


Figura 7 Formulario introducción de pedidos

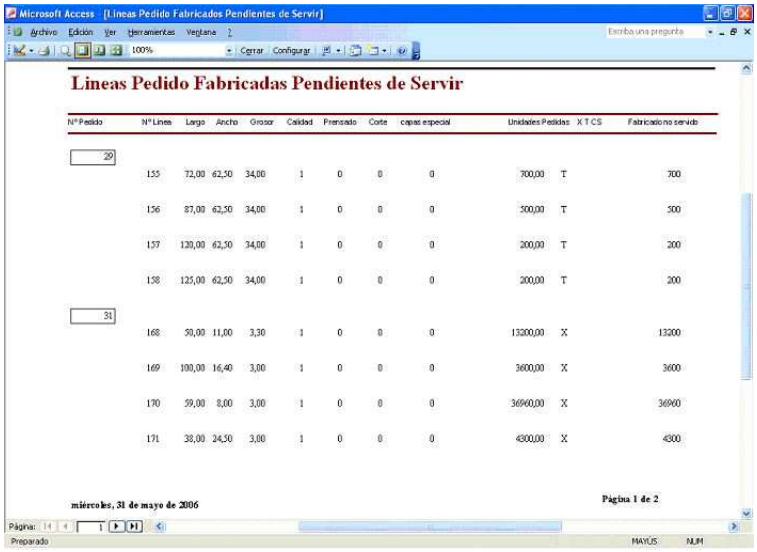


Figura 8 Informe líneas de pedido fabricadas y pendientes de servir

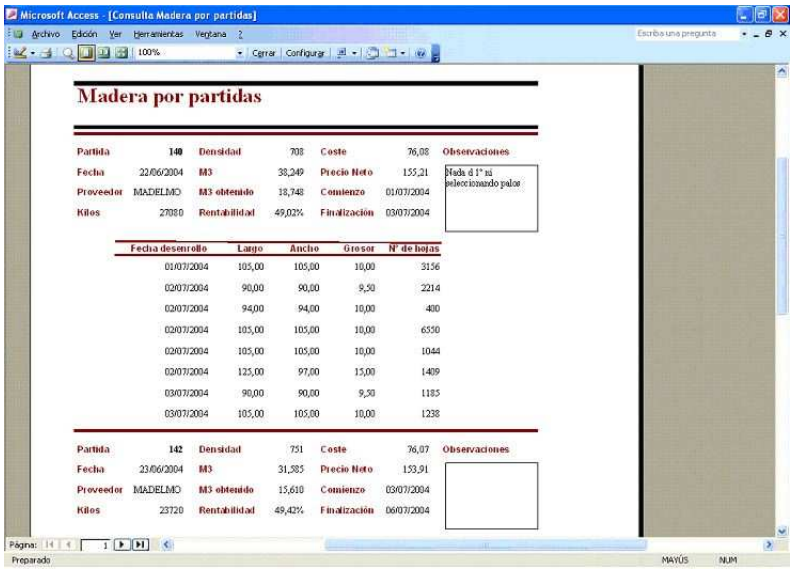


Figura 9. Informe sobre el rendimiento de la madera que se ha obtenido en cada partida

Por último, la figura 10 muestra el informe que recoge la solución proporcionada por el programa de planificación. En concreto, aparece toda la información

asociada al lote que ocupa el primer lugar en la solución propuesta: número de tableros, características, hojas necesarias de cada calidad, pedidos asociados, número de cortes en la retestadora, tiempo de ejecución en prensa y corte, etc.

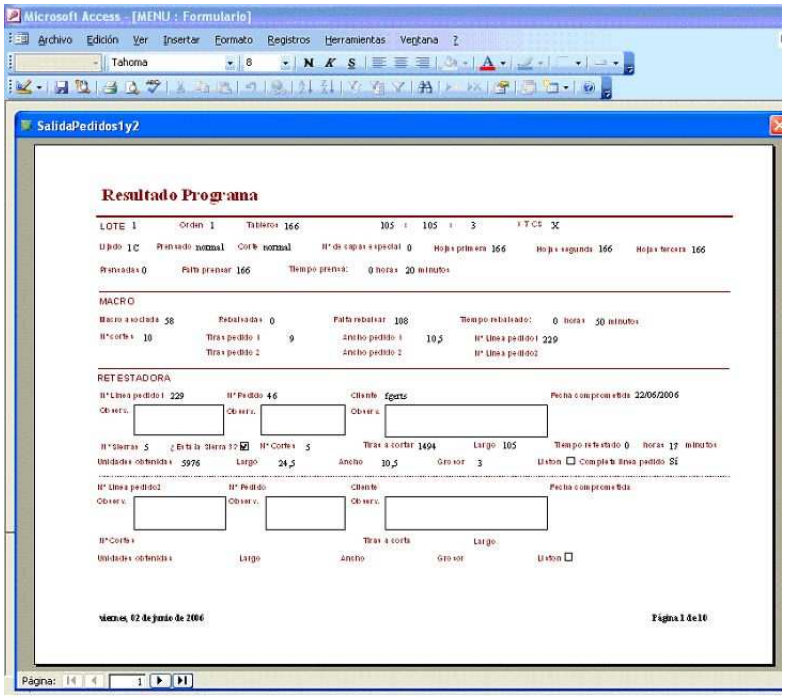


Figura 10. Salida del programa

6 Conclusiones

En este trabajo se ha analizado el sistema productivo de una empresa de contrachapado y se ha propuesto un sistema de información de ayuda a la gestión. Este sistema se ha diseñado ex profeso para la situación que define a la empresa, desarrollando algoritmos para la parte del sistema productivo detectada como determinista y susceptible de ser automatizada. Los algoritmos forman parte del sistema de planificación que se comunican con la base de datos, donde se almacenan todos los datos de la empresa. La información necesaria para planificar es suministrada por la base de datos. Asimismo, la información producida por los algoritmos de planificación también se almacena en la base de datos. Así la información global permite al gestor planificar, de manera más eficiente, la parte del sistema productivo no susceptible de ser automatizada.

Los beneficios obtenidos al planificar la sección de corte y propagar esta información hacia atrás hasta la sección de desenrollo han sido muy importantes: disminución del número de los pedidos que se convierten en urgentes por una inadecuada planificación; consecuentemente, disminución del número de horas extraordinarias; mayor utilización de la retestadora lo que implica aumentar la capacidad de producción de la empresa; disminución de desperdicios...

Además, para el buen funcionamiento del sistema, es preciso que los datos estén actualizados en su globalidad, lo que implica la posibilidad de obtener información válida en todo momento. Todo ello facilita la elaboración de informes para los encargados de las distintas secciones y de los documentos administrativos habitualmente necesarios.

7 Bibliografía

[1] Aldowaisan, T., Allahverdi, A. and Gupta, J. N. (1999). A review of scheduling research involving setup considerations. Omega, 27: 219 – 239.

[2] Ballestín, F., Valls, V. and Quintanilla, S. (2006). Due dates and RCPSP. In J. Jozefowska and J. Weglarz, editors, Perspectives in Modern Project Scheduling. Kluwer.

[3] Du, J. and Leung., J. Y.–T. (1990). Minimizing total tardiness on one machine is NP-hard. Mathematics of Operations Research 15, 483 - 495.

[4] Feo, T. A. and Resende, M. G. C. (1995). Greedy randomized adaptive search procedures. Journal of Global Optimization, 6: 109 - 133.

[5] Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning, Addison – Wesley, Reading, Massachusetts.

[6] Hartmann, S. (1998). A competitive genetic algorithm for resource-constrained project scheduling. Naval Research Logistics, 45: 733 - 750.

[7] Holland, J. H. (1975). Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence, (MIT Press: Michigan).

[8] Trautmann, N. (2001). Anlagenbelegungsplannung in der Prozessindustrie. Gabler, Wiesbaden.

[9] Valls, V., Ballestín, F. y Quintanilla, S. (2002). A hybrid genetic algorithm for the RCPSP. Technical Report 8-01, Departamento de Estadística e Investigación Operativa, Universitat de València.

Mejorando las soluciones de un *Strip Packing Problem.* Método de mejora dependiente del problema*

José Ignacio García del Amo[a], **J. Marcos Moreno-Vega**[a]

[a] Dpto. de Estadística, I.O. y Computación,
Escuela Técnica Superior de Ingeniería Informática
Universidad de La Laguna
igdelamo@ull.es, jmmoreno@ull.es

1 Introducción

Muchas técnicas heurísticas de resolución de problemas aplican procedimientos de mejora a soluciones previamente obtenidas. Los procedimientos de mejora más conocidos son las Búsquedas Locales en las que, una vez definido el concepto de entorno de una solución, se escoge una solución del mismo que mejore a la solución inicial. Si esta solución existe, el procedimiento reitera el paso anterior con ella. En caso contrario, se finaliza la búsqueda. La gran mayoría de los procedimientos de mejora que pueden aplicarse a una solución son variantes de las Búsquedas Locales o emplean a éstas como elementos importantes de su diseño.

Frecuentemente, los métodos de mejora son independientes de la solución que se desea mejorar. Así, se aplican por igual a buenas y a malas soluciones. No realizan un análisis de las soluciones que les permita adaptarse a las características

*Este trabajo ha sido parcialmente financiado por el Ministerio de Ciencia y Tecnología (proyecto TIN2005-08404-C04-03 (70% son fondos FEDER)) y por el Gobierno de Canarias (proyecto PI042004/088). La actividad desarrollada se enmarca dentro de los objetivos de la red RedHeur (proyecto TIN2004-20061-E).

de las mismas. Como consecuencia, emplean una mayor cantidad de recursos computacionales. El análisis de las soluciones permite diseñar métodos de mejora más eficientes y eficaces.

Presentamos un procedimiento alternativo para mejorar la calidad de una solución del problema del empaquetado rectangular bidimensional no guillotina (*Strip Packing Problem*). El procedimiento tiene su origen en el análisis de una técnica constructiva GRASP [1] previamente propuesta para este problema. Se ha observado que esta técnica tiende a ubicar de forma incorrecta los rectángulos en las últimas iteraciones del método constructivo. Por ello, se propone reconstruir la solución obtenida por el método constructivo extrayendo los últimos rectángulos de la solución para ubicarlos usando una técnica heurística. Realizamos un análisis de la solución obtenida por el método constructivo para determinar qué rectángulos deben extraerse de la solución. Con ello, adaptamos el procedimiento de mejora al problema y así aumentamos la eficiencia del mismo.

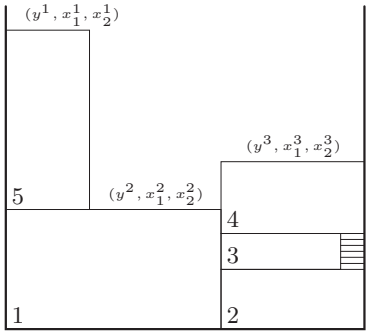
El presente trabajo se estructura de la siguiente forma. En la próxima sección se introduce el *Strip Packing Problem*. En la sección 3 se expone la técnica constructiva GRASP y en la sección 4 se describe la particular implementación que se hace de la misma para el *Strip Packing Problem*. Además, se describe el análisis de la solución que da lugar al procedimiento de mejora. Por último, se muestra la experiencia computacional realizada y se enumeran las conclusiones que se siguen de los resultados obtenidos.

2 *Strip Packing Problem*

Los problemas de empaquetado constituyen una amplia clase de problemas en los que, de forma general, se desea empaquetar un conjunto de items (figuras geométricas pequeñas) en un objeto geométrico mayor (o conjunto de objetos) de tal forma que se optimice algún objetivo relativo al empaquetado obtenido.

La importancia de estos problemas en procesos industriales o de gestión financiera se refleja en la gran cantidad de trabajos aparecidos en la literatura científica. Algunos trabajos de revisión y clasificación en los que también se enumeran aplicaciones son [4] [5] [6] [10] y [11].

Aquí consideramos el *Strip Packing Problem* que se formula como sigue. Dado un objeto rectangular de amplitud fija w y altura infinita, y un conjunto, $\mathcal{R} = \{R(w_1, h_1), \ldots, R(w_n, h_n)\}$, de rectángulos con al menos uno de sus lados, w_i, h_i, menor que w, se desea empaquetar el conjunto $\mathcal{R}$ en el objeto rectangular utilizando el menor espacio posible (o lo que es lo mismo, se pretende minimizar la altura del empaquetado). En este problema se pueden rotar los objetos y los cortes pueden ser de tipo no guillotina. Un corte es tipo guillotina si atraviesa el objeto desde un lado hasta el lado opuesto. En un corte no guillotina, lo anterior no es cierto. En la figura 1(a)) se muestra una solución de un *Strip Packing Problem*.



(a) Solución

(b) Contorno superior

3 Greedy Randomized Adaptive Search Procedures

En un método constructivo se añade iterativamente elementos a una estructura, inicialmente vacía, hasta obtener una solución del problema. La elección del elemento a incluir se basa en una evaluación *heurística*, que mide la conveniencia de considerar este elemento como parte de la solución. La función heurística es dependiente del problema y expresa el conocimiento que sobre el mismo se tiene. Si la evaluación de un elemento depende de los elementos previamente incluidos en la solución se dice que el método es *adaptativo*.

Además de la función heurística, es necesaria una estrategia que indique qué elemento se escoge. Una de las estrategias más conocidas es la *greedy* en la que se selecciona el elemento que optimiza la función heurística. Esta estrategia suele dar pobres resultados en la mayoría de los casos. Por ello se han propuesto estrategias alternativas. Una de ellas consiste en elegir, no el mejor elemento, sino uno de los mejores al azar. Al conjunto de los mejores elementos se le llama *Lista Restringida de Candidatos* (LRC).

GRASP (Greedy Randomized Adaptive Search Procedure) [7][8][9] es un procedimiento heurístico que consta de varias etapas. A una fase constructiva, en la que se escoge iterativamente y al azar un elemento de la lista restringida de candidatos, le sigue una fase de postprocesamiento en la que se mejora la solución obtenida en la fase anterior. Como postprocesamiento suele emplearse una simple búsqueda local descendente. Los anteriores pasos se reiteran hasta que se cumpla el criterio de parada. La mejor solución obtenida es la propuesta por el algoritmo. En ocasiones, se considera una fase de preprocesamiento previa a la fase constructiva. El propósito de esta fase es acelerar la fase constructiva posterior, incluyendo aquellos elementos que, en base a algún criterio, *deben* estar en

la solución. Así, pueden incluirse aquellos elementos que necesariamente pertenecen a la solución óptima del problema, o aquellos elementos que, en base a la experiencia del decisor o historia pasada de la búsqueda, pertenecen a soluciones de alta calidad. En [3] se ha propuesto un GRASP para un problema de ruta de vehículos en el que el usuario incorpora conocimiento al procedimiento en la forma de soluciones en las que ciertos clientes deben ser servidos por vehículos específicos. En la figura 1 se muestra un pseudocódigo del GRASP.

```
procedure GRASP;
begin
   Solución Inicial := Fase de preprocesamiento;
   Solución Actual := Solución Inicial;
   Mejor Solución := Solución Actual;
   repeat
      Solución Actual := Fase Constructiva;
      Solución Actual := Fase de Postprocesamiento
      If Objetivo(Solución Actual) <
            Objetivo(Mejor Solución)
      then Mejor Solución := Solución Actual;
   until (Criterio de parada);
end
```

Figura 1: Pseudocódigo descriptivo del GRASP.

Los elementos que determinan completamente la técnica GRASP son: el método de preprocesamiento, la función heurística, la forma en que se construye la lista restringida de candidatos, el método de postprocesamiento y el criterio de parada. De los anteriores elementos, algunos son totalmente dependientes del problema y para otros pueden hacerse elecciones dependientes o independientes del mismo. Así, la función heurística es dependiente del problema, y la regla de parada puede ser dependiente o independiente del problema. Algunas reglas de parada independientes del problema son finalizar la búsqueda si se alcanza un número máximo de iteraciones o se sobrepasa el tiempo máximo de CPU previamente establecido. En [2] se proponen reglas de parada dependientes del problema que analizan las características de las soluciones obtenidas para decidir cuando finalizar la búsqueda.

4 GRASP para el *Strip Packing Problem*

En la presente sección describimos un GRASP para el *Strip Packing Problem*. En [1] se encuentra una descripión detallada de este GRASP junto al análisis de

amplios resultados experimentales que muestran la eficiencia y eficacia de esta técnica.

La definición de la lista restringida de candidatos se realiza a partir del concepto de contorno superior.

4.1 Contorno superior

La inclusión de un rectángulo cualquiera en el objeto, determina un contorno superior rectangular como el que se muestra en la figura 1(b). Además, es posible que se obtengan áreas no aprovechables, llamadas desperdicios, como el que se obtiene al incluir el rectángulo 4 en el objeto de la figura 1(b). El contorno, C, puede representarse por medio del conjunto de segmentos horizontales (tomados de izquierda a derecha) que lo forman. Es decir:

$$C = \{(y^1, x_1^1, x_2^1), (y^2, x_1^2, x_2^2), \ldots, (y^c, x_1^c, x_2^c)\}$$

con

$$\begin{array}{rcl} y^i & \equiv & \text{altura del } i\text{-ésimo segmento} \\ x_1^i & \equiv & \text{punto inicial del } i\text{-ésimo segmento} \\ x_2^i & \equiv & \text{punto final del } i\text{-ésimo segmento} \end{array} .$$

Además, en el primer contorno $x_1^1 = 0$ y $x_2^c = w$. Nótese que, intuitivamente, es preferible un contorno formado por pocos segmentos. Esto es así, ya que, en general, la posibilidad de obtener desperdicios aumenta con el número de segmentos.

4.2 Lista restringida de candidatos

Sea t la iteración actual del proceso constructivo y supongamos que $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$, siendo $\mathcal{R}_1$ el conjunto de los rectángulos previamente incluidos en el objeto y $\mathcal{R}_2 = \mathcal{R} \setminus \mathcal{R}_1$. Sea $C(t)$ el contorno determinado por los rectángulos de $\mathcal{R}_1$. Evaluaremos la conveniencia de incluir un rectángulo de $\mathcal{R}_2$ en el objeto por la forma que tendrá el contorno $C(t)$ tras su inclusión.

> *Lista restringida de candidatos:* sea dado $\alpha \in [0, 1]$ y supongamos que el segmento del contorno con menor altura es (y^i, x_1^i, x_2^i). La lista restringida de candidatos se construye como sigue:
>
> $$LRC = \{R(w_j, h_j) \in \mathcal{R}_2 : (0 \leq x_2^i - x_1^i - w_j \leq \alpha) \vee (0 \leq x_2^i - x_1^i - h_j \leq \alpha)\}.$$
>
> Es decir, la lista está formada por aquellos rectángulos que mejor se ajustan al ancho del segmento inferior del contorno. El ajuste viene determinado por el valor de α.

Para que la anterior definición tenga sentido, debe haber, al menos, un rectángulo de $\mathcal{R}_2$, digamos $R(w_r, h_r)$, tal que $(0 \leq x_2^i - x_1^i - w_r \leq \alpha) \vee (0 \leq x_2^i - x_1^i - h_r \leq \alpha)$. Si ningún elemento de $\mathcal{R}_2$ cumple la anterior condición, se escoge el rectángulo que mejor se ajusta a $x_2^i - x_1^i$, y se reconstruye el contorno. Si no existe tal rectángulo, se reconstruye el contorno eliminando, convenientemente, el segmento (y^i, x_1^i, x_2^i).

4.3 Fase de postprocesamiento

Una de las situaciones anómalas que puede presentarse al aplicar los métodos constructivos anteriores se muestra en la figura 2. Consideremos la ubicación del rectángulo 6. Cualquiera de los métodos anteriores lo ubicaría según se indica en la figura 2(b). La bondad de esta nueva situación depende del instante en que se produce. En las primeras iteraciones del método, la situación es aconsejable. No obstante, en las últimas iteraciones puede producir soluciones de baja calidad. En particular, si nos encontramos en la última iteración, sería preferible ubicarlo como se muestra en la figura 2(c).

El anterior comportamiento es característico del método constructivo que empleamos. Por ello, en [1] se proponía el procedimiento de mejora consistente en extraer los últimos rectángulos de la solución y ubicarlos de la mejor manera posible. Para ello, se consideran todas las ordenaciones posibles de los rectángulos extraídos y, para cada una de ellas, se colocan, según el orden establecido en la ordenación, los rectángulos en la posición, y con la orientación, que alcanza una menor altura relativa.

A pesar de que la anterior fase de postprocesamiento mejora la calidad de las soluciones obtenidas en la fase constructiva (ver [1]), tiene dos inconvenientes.

1. *Valores limitados del parámetro m:* dado que se realiza una búsqueda exhaustiva entre todas las posibles combinaciones de los últimos m rectángulos, hay que limitar esta búsqueda a valores peque nos de m.

2. *Obligación de fijar a priori el valor de m:* soluciones diferentes requerirán, posiblemente, reconstruir desde puntos distintos. Fijar a priori el valor de m impide que la fase de postprocesamiento se adapte a la solución obtenida en la fase constructiva.

Para subsanar estos inconvenientes, se propone analizar el contorno superior que se obtiene en cada iteración del proceso constructivo y determinar de esta forma el valor de m, y realizar una búsqueda heurística entre todas las combinaciones de los últimos m rectángulos.

Análisis del contorno

La fase constructiva de un GRASP para el problema del empaquetado de rectángulos bidimensional no guillotina consta de n iteraciones (con n el número

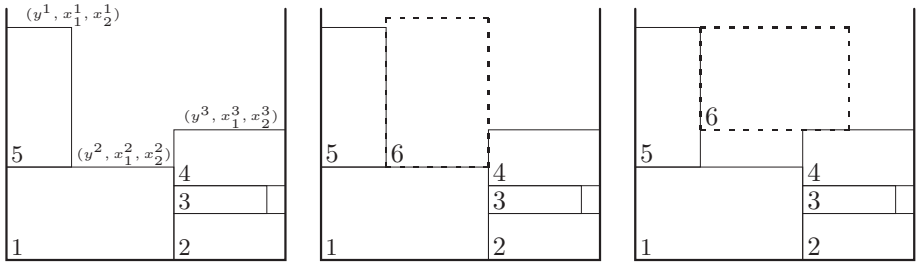


Figura 2: Procedimiento de mejora

de rectángulos a empaquetar). Sea $C(t)$ el contorno superior que se obtiene al incluir un rectángulo en la t-ésima iteración.

Asociado al contorno $C(t)$ pueden considerarse varios valores que miden la suavidad del mismo. Uno de estos valores es la altura media de los segmentos que se define como

$$AlturaMedia(C(t)) = \frac{1}{c}\sum_{i=1}^{c}(y^+(C(t)) - y^i) \qquad t = 1, \ldots, n$$

donde c es el número de segmentos del contorno y

$$y^+(C(t)) = \max_{i=1,\ldots,c}\{y^i\} \qquad t = 1, \ldots, n.$$

Convenimos que $AlturaMedia(C(0)) = 0$. Sea asimismo

$$\Delta AlturaMedia(t) = AlturaMedia(C(t-1)) - AlturaMedia(C(t)),\ t = 1, \ldots, n$$

el incremento que se produce en la altura media del contorno al incluir el rectángulo de la t-ésima iteración.

Proponemos el siguiente método para obtener la iteración (o equivalentemente el número de rectángulos) a partir de la cual aplicar el método de mejora.

> *Mayor incremento en la altura media.* Aplicar el procedimiento de mejora a partir de la iteración en que se produce un mayor incremento en la altura media. Es decir, si t^* es la iteración que maximiza
>
> $$\Delta AlturaMedia(t), \qquad t = 1, \ldots, n,$$
>
> se extraen los rectángulos empaquetados en las iteraciones que van desde la t^* hasta la n. Es decir, se extrae aquel rectángulo que, en base al criterio anterior, ha sido mal colocado (ha producido un mayor incremento en la altura media) y todos los que han sido colocados posteriormente.

Búsqueda heurística

Nótese que una vez determinada la iteración, t^*, a partir de la cual reconstruir, se obtiene un nuevo problema de empaquetado de dimensión menor en el que la frontera inferior del objeto en que se deben incluir los rectángulos viene dada por el contorno $C(t^*)$. Este nuevo problema puede abordarse por cualquiera de las técnicas heurísticas de resolución de problemas. Hemos experimentado con una Búsqueda Local Descendente.

> *Búsqueda Local Descendente.* Dada una permutación de los rectángulos y el movimiento consistente en intercambiar el orden de dos ellos, realizar el mejor de los movimientos mientras sea posible. Los rectángulos se empaquetan, según el orden establecido en la ordenación, en la posición, y con la orientación, que alcanza una menor altura relativa.

4.4 Criterio de parada

La búsqueda realizada con GRASP finaliza después de un número dado, $n_{iter} = 20$, de pasadas del bucle *Fase Constructiva*, *Fase de Postprocesamiento*.

5 Experiencia computacional

Para evaluar el comportamiento de la propuestas de mejora de las soluciones obtenidas en la fase constructiva del GRASP, se resolvieron diferentes problemas generados aleatoriamente. Se implementó un generador de problemas que, dado el ancho del objeto rectangular, w, el número de rectángulos, n, y el valor objetivo óptimo, h_{opt}, suministra un conjunto de n rectángulos que pueden ubicarse en un objeto rectangular de amplitud w utilizando una altura h_{opt}. En la figura 1(a) se muestra uno de los problemas obtenidos con este generador. Cada problema fué resuelto 5 veces realizando 20 pasadas del bucle *Fase Constructiva*, *Fase de Postprocesamiento*. El valor que determina el umbral de ajuste en el proceso constructivo del GRASP se fijó a $\alpha = 0$.

En la tabla 1 se muestran los resultados obtenidos en la experiencia computacional. Las tres primeras columnas describen el problema: número de rectángulos (n), ancho del objeto rectangular (w) y altura óptima (h_{opt}).

En las columnas 4 y 5 se recogen el mejor valor objetivo (Obj) y el valor objetivo medio obtenido en las 20 fases constructivas del GRASP. En las columnas 6 y 8 se muestran estos mismos valores tras la fase de postprocesamiento. Las columnas 7 y 9 almacenan las mejoras, respecto de la fase constructiva, producidas tras la fase de postprocesamiento.

Por cada problema se muestran 5 filas de resultados (una por cada una de las 5 ejecuciones del GRASP realizadas) más una que recoge los valores medios de las mejoras. De los resultados obtenidos podemos concluir lo siguiente.

Tabla 1: Evaluación experimental del postprocesamiento

			Constr.		Postproc.			
n	w	h_{opt}	Obj	$\overline{Obj}$	Obj	$Mej.$	$\overline{Obj}$	$Mej.$
50	50	50	51	52.6	51	0	52.45	0.15
			52	53.15	51	1	52.95	0.2
			52	55.7	51	1	55.15	0.55
			52	55.2	51	1	54.9	0.3
			52	52.55	52	0	52.45	0.10
						0.6		0.26
50	40	60	64	64.2	62	2	63.75	0.45
			62	65.55	62	0	63.15	0.85
			64	66.3	63	1	64.9	1.4
			64	65.9	64	0	64.4	0.4
			63	65.45	63	0	64.45	1
						0.6		0.82
100	50	50	52	52.55	52	0	52.55	0
			52	52	51	1	51.35	0.65
			52	52	52	0	52	0
			51	51.65	51	0	51.65	0
			52	52	51	1	51.35	0.65
						0.4		0.26
100	50	75	77	77.35	77	0	77.05	0.3
			76	76.75	76	0	76.75	0
			77	77.3	77	0	77	0.3
			77	77	77	0	77	0
			77	77.55	77	0	77.25	0.3
						0		0.18
200	100	100	101	101.8	101	0	101.8	0
			101	101.8	101	0	101.5	0.3
			102	102.2	101	1	102.05	0.15
			101	103.85	101	0	102.65	1.2
			101	102.2	101	0	102.05	0.15
						0.2		0.36
200	120	160	163	164.75	162	1	163.7	1.05
			163	166.9	162	1	165.3	1.6
			162	169.25	162	0	169.25	0
			163	165.55	162	1	165.05	0.5
			165	167.1	163	2	165.75	1.35
						1		0.9

1. *La fase de postprocesamiento mejora la calidad de las soluciones.* La mejora se produce tanto en el mejor valor objetivo como en el valor objetivo medio. Aunque la mejora pueda parecer no significativa hay que señalar que las soluciones generadas en la fase constructiva son, en la gran mayoría de los casos, de alta calidad. Por ello, el efecto de la mejora puede parecer menor.

2. *La técnica GRASP propuesta es eficaz en la resolución del problema.* La distancia que existe entre el mejor valor objetivo encontrado por GRASP y el valor objetivo óptimo es inferior en todos lo casos a 2 unidades (para cuatro problemas es inferior a 1 unidad y para los otros dos es inferior a 2 unidades). Este comportamiento ya se observó en la amplia experiencia computacional desarrollada en [1].

6 Bibliografía

[1] Beltrán, J. D, Eduardo, J., Jorge, R., Moreno-Vega, J.M. Procedimientos constructivos adaptativos (GRASP) para el problema del empaquetado bidimensional *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial* **15** (2002) 26–33

[2] Beltrán, J. D, Eduardo, J., Jorge, R., Moreno-Vega, J.M. Reglas de parada para el problema del empaquetado rectangular bidimensional no guillotina. Actas de la VIII Conferencia Iberoamericana de Inteligencia Artificial. IBERAMIA 2002 pp. 107-116, Sevilla. Del 12 al 15 de Noviembre de 2002

[3] Carlos Carreto, Barrie Baker *A Grasp Interactive Approach to the Vehicle Routing Problem with Backhauls* pp. 185-199. En Essays and Surveys in Metaheuristics. Celso C. Ribeiro, Pierre Hansen (Eds) Kluwer Academic Publishers, 2002

[4] Dowsland, K.A., Dowsland, W.B. Packing Problems *European Journal of Operational Research* **56** (1992) 2–14

[5] Dyckhoff, H. Typology of Cutting and Packing Problems. *European Journal of Operational Research* **44** (1990) 145–159

[6] Dyckhoff, H., Finke, U. *Cutting and Packing in Production and Distribution.* Spronger Verlag, Berlin (1992)

[7] Feo, T. A., Resende, M.G.C. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters* **8** (1989) 67–71

[8] Feo, T. A., Resende, M.G.C. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization* **6** (1995) 109–133

[9] P. Festa, M.G.C. Resende *GRASP: An Annotated Bibliography* pp. 325–367. En Essays and Surveys in Metaheuristics. Celso C. Ribeiro, Pierre Hansen (Eds) Kluwer Academic Publishers, 2002

[10] Hopper, E., Turton, B.C.H. A Review of the Application of Meta-Heuristics Algorithms to 2D Strip Packing Problems. *Artificial Intelligence Review* **16** (2001) 257–300

[11] Wäscher, G., Haussner, H., Schumann, H.: An improved typology of cutting and packing problems *European Journal of Operational Research*, En prensa (2006)

Utilizando Búsqueda Tabú para el diseño de una red multiproducto con capacidades en las aristas

A.M. Alvarez Socarrás[a], **F. Angel-Bello**[b], **N. Cobos Zaleta**[a]

[a] Universidad Autónoma de Nuevo León
[b] Instituto Tecnológico y de Estudios Superiores de Monterrey,

1 Introducción

El problema de diseño de red óptima se caracteriza por la búsqueda de la mejor configuración de la red que satisfaga un conjunto dado de requerimientos. Este problema ha sido estudiado por un buen número de investigadores ya que su solución es relevante en numerosas aplicaciones y se conoce que pertenece a la clase NP- completo. Actualmente no existen algoritmos exactos que puedan resolver problemas similares en un período de tiempo razonable, especialmente para instancias grandes. De ahí que otro tipo de técnicas, tales como las heurísticas, deban emplearse, si bien no para encontrar la solución óptima, sí para encontrar soluciones de calidad aceptable.

Este tipo de problemas se presentan en muy diversos campos, por ejemplo en diseños, re-diseños de redes de computadoras o telecomunicación, distribución, sistemas eléctricos de potencia, de redes de transporte, etc. Una amplia compilación de modelos y aplicaciones puede ser encontrada en la obra de Magnanti y Wong [10].

En el presente trabajo abordamos el siguiente problema de diseño de redes: dado un conjunto de nodos y un conjunto de aristas potenciales, se deben seleccionar las aristas que formarán parte de la red, de modo que puedan circular por

ella diversos productos entre ciertos pares origen-destino establecidos, sin exceder la capacidad de cada arista. Al hacer esto se incurre en dos diferentes tipos de costos: los costos variables por transportar una unidad de producto y los costos fijos pagados por la inclusión de las aristas en el diseño de la red. El objetivo es determinar qué aristas deben considerarse en el diseño de forma que se garantice la operación de la red y que el costo total en que se incurra (considerando costos de diseño y de operación) sea el menor posible.

Se puede encontrar una extensa bibliografía en la versión no capacitada del problema, pero el caso que considera capacidades finitas en las conexiones no se encuentra en igual situación. Este tipo de problemas es más realista, pero también mucho más complejo, y ha sido abordado con diferentes metodologías tales como relajación lagrangeana [2], procedimiento de acotamiento [3], Tabu Search [9], etc.

Sin embargo estos trabajos han sido desarrollados para redes orientadas. Esto significa que en esas redes puede que entre dos nodos sólo exista un arco en un solo sentido mientras que en el presente trabajo, siempre que se considere conexión entre dos nodos se considerarán los arcos en los dos sentidos. Además, en esos trabajos los arcos dirigidos que unen dos nodos poseen capacidades independientes, una para cada sentido del arco, mientras que en el problema aquí abordado, existe una única capacidad para la arista que une dos nodos, es decir, esta capacidad debe ser compartida por todos los productos que circulen por los dos arcos orientados correspondientes a la arista, sin importar el sentido del flujo. Estas características impiden cualquier adaptación inmediata de los métodos propuestos para redes orientadas al problema aquí abordado.

Para redes no orientadas solo se han reportado resultados obtenidos con el diseño de una metaheurística evolutiva, los cuales no son satisfactorios para redes con ciertas características [1] y otro trabajo desarrollado por Herrmann [7] que utiliza un método de ascenso dual para redes tipo malla.

Por todo lo anterior y teniendo en cuenta que Tabú Search resulto efectivo en un problema similar, pero sobre redes orentadas [9], se ha considerado importante estudiar la efectividad en este problema de dicha metaheurística, caracterizada por una exploración inteligente de estructuras de memoria.

2 Planteamiento del problema

Sea $G = (N, A)$ un grafo que representa una red no orientada con un conjunto N de nodos y un conjunto A de aristas potenciales. Sea A' el conjunto de arcos potenciales asociados a esas aristas. Sea K el conjunto de productos con demanda d^k para el k-ésimo producto. Del conjunto de nodos se distinguirán varias parejas origen-destino, asociadas cada una de ellas a un producto que circulará por la red. Sean $O(k)$ y $D(k)$ el nodo origen y nodo destino respectivamente del k-ésimo producto.

Por otra parte, a cada arista potencial $\{i,j\}$ se le asigna un costo fijo F_{ij} por su utilización o construcción, así como costos c_{ij}^k y c_{ji}^k por unidad de producto transportado, el cual depende del tipo de producto de que se trate y del sentido en que circule por la arista. Cada arista $\{i,j\}$ tiene asociada una capacidad finita u_{ij}, la cual deberá ser compartida por todos los productos que circulen en cualquier dirección de la misma.

El modelo tiene dos tipos de variables de decisión. El primer tipo es una variable binaria que modela las elecciones de diseño y se define como $y_{ij} = 1$, si la arista $\{i,j\}$ se incluye en el diseño de la red, o bien, $y_{ij} = 0$, en caso contrario. El segundo tipo es una variable continua que modela las decisiones de flujo del producto k que circula por el arco orientado (i,j) y que denotaremos como x_{ij}^k.

Debe señalarse que, una vez que se haya decidido conectar dos nodos i,j (esto es, la arista$\{i,j\}$ formará parte del diseño de la red), se permitirá flujo en ambos sentidos, o sea, se considerarán en la red ambos arcos (i,j) y (j,i).

Para cada producto $k \in K$ y cada $i \in N$ impondremos las restricciones usuales de conservación de flujo en redes

$$\sum_{\{j:(i,j)\in A'\}} x_{ij}^k - \sum_{\{j:(j,i)\in A'\}} x_{ji}^k = \begin{cases} d^k & \text{si } i = O(k) \\ -d^k & \text{si } i = D(k)\ \forall k \in K, \\ 0 & \text{en caso contrario } \forall i \in N \end{cases} \quad (1)$$

También consideraremos que el flujo de todos los productos que circulan en cualquier dirección por cada arista $\{i,j\}$ no debe exceder la capacidad de dicha arista.

$$\sum_{k\in K} (x_{ij}^k + x_{ji}^k) \le u_{ij} y_{ij} \qquad \forall \{i,j\} \in A \quad (2)$$

Estas restricciones no solo aseguran que sean respetadas las capacidades de las aristas, sino también fuerzan a que el flujo de cualquier producto x_{ij}^k sea cero si la arista $\{i,j\}$ no ha sido seleccionada en el diseño. Por último, exigimos que las variables continuas sean no negativas y que las variables de diseño sean binarias.

$$\begin{array}{ll} x_{ij}^k \ge 0 & \forall k \in K;\ \forall (i,j) \in A' \quad (3) \\ y_{ij} \in \{0,1\} & \forall \{i,j\} \in A \end{array}$$

Nuestro objetivo será minimizar el costo total en que se incurre por diseño y operación de la red, esto es,

$$\min \left[\sum_{k\in K} \sum_{(i,j)\in A'} c_{ij}^k x_{ij}^k + \sum_{\{i,j\}\in A} F_{ij} y_{ij} \right] \quad (4)$$

Al problema (1)-(4) lo referiremos como DRCM.

Las decisiones a tomar consisten en la selección de las aristas que deben incluirse en el diseño final de la red esto es, los valores de las variables y_{ij} y los

volúmenes de flujo de cada producto que circularán por cada una de las aristas incluidas en sus dos direcciones (x_{ij}^k, x_{ji}^k) para satisfacer las demandas.

Es importante destacar aquí dos aspectos esenciales del problema:

1. Una vez que una arista ha sido incluida en el diseño de la red, se permitirá flujo en ambas direcciones, por lo que los costos variables de transportación asociados a cada arista dependerán no solamente del producto sino también del sentido en que esté circulando.

2. Cada arista posee una capacidad finita que será compartida por todos los productos que la usen sin importar la dirección del flujo para esos productos.

La gran dificultad de este problema estriba en dos cosas: un equilibrio entre costos fijos y variables al construir una solución, y una interacción entre los diferentes productos que comparten las capacidades de cada arista en la red.

3 Tabu Search

Los orígenes de Búsqueda Tabú (Tabú Search, TS, en inglés) pueden situarse en diversos trabajos publicados hace alrededor de 20 años. Oficialmente, el nombre y la metodología fueron introducidos posteriormente por Fred Glover [6]. Numerosas aplicaciones han aparecido en la literatura, así como artículos y libros para difundir el conocimiento teórico del procedimiento [5].

TS es un procedimiento metaheurístico para resolver problemas de optimización combinatoria, utilizado para guiar cualquier procedimiento de búsqueda local, en la exploración del espacio de soluciones más allá de la simple optimalidad local.

TS se basa en la premisa de que para poder calificar de inteligente la resolución de un problema, debe incorporar memoria adaptativa y exploración sensible, que son las características principales de búsqueda tabú. El éxito de esta metodología en diversos problemas se debe a sus estructuras de memoria y al uso de estrategias de intensificación y diversificación. Las estrategias de memoria evitan retornar a soluciones visitadas anteriormente, permiten guardar atributos de buenas soluciones contribuyendo a identificar regiones de interés y más generalmente guiar la exploración del espacio de solución. Las estrategias de intensificación y diversificación permiten avanzar a una solución vecina que es peor que la solución actual, pero que proporciona la posibilidad de penetrar a un espacio del conjunto de soluciones factibles que de otro modo no habría sido visitado y que podría contener una solución óptima global al problema.

Tabú Search distingue dos tipos de memoria: a corto y a largo plazo.

- La memoria a corto plazo, está *basada en atributos,* es decir almacena atributos de soluciones recientemente visitadas y su objetivo es explorar a fondo

una región dada del espacio de soluciones. En ocasiones se utilizan estrategias de listas de candidatos para restringir el número de soluciones examinadas en una iteración dada o para mantener un carácter agresivo en la búsqueda.

- La memoria a largo plazo almacena las frecuencias u ocurrencias de atributos en las soluciones visitadas tratando de identificar o diferenciar regiones. Estas tienen dos estrategias asociadas: *Intensificar* y *Diversificar la búsqueda.* La intensificación consiste en regresar a regiones ya exploradas para estudiarlas más a fondo. Para ello se favorece la aparición de aquellos atributos asociados a buenas soluciones encontradas previamente. La diversificación consiste en visitar nuevas áreas no exploradas del espacio de soluciones. Para ello se modifican las reglas de elección para incorporar a las soluciones atributos que no han sido usados frecuentemente.

4 Metodología de Solución

El procedimiento diseñado, el cual está inspirado en un método propuesto por Crainic et al. para redes orinetadas [9], puede resumirse en los siguientes pasos que explicaremos más adelante.

1. Obtener una solución inical.
2. Realizar una búsqueda local.
3. Ejecutar movimientos de diversifiación.
4. Repetir pasos 2 y 3 un número predefinido de veces.

4.1 Obtención de la solución inicial

La solución inicial (X, Y) donde

$$X = \{x_{ij}^k \ \forall k \in K, \forall (i,j) \in A'\}, \quad Y = \{y_{ij} \ \forall \{i,j\} \in A\}$$

se obtiene mediante un procedimiento greedy aleatorizado desarrollado a tal efecto. El procedimiento que se usa está basado en la técnica heurística del GRASP [8]. Por consiguiente, esta solución inicial ya es relativamente buena.

4.2 Búsqueda local

Dada una solución factible (X, Y) es necesario definir la vecindad donde se realizará la búsqueda local. Esta vecindad, la cual denominaremos vecindad continua, consta de todas las soluciones que pueden alcanzarse a partir de la actual mediante un pivoteo simplex.

Consecuentemente, un movimiento local corresponde a una transición de una base del sistema (1) - (3) a una adyacente, o sea, un camino básico es sustituido por uno de los actualmente no básicos.

Para determinar qué movimiento implementar, se determina, para cada posible camino a entrar a la base, el correspondiente camino básico a salir y se evalúa el "valor" de este movimiento potencial (en cuanto a la mejoría de la función objetivo). Se selecciona e implementa el movimiento "mejor" si no es tabu, o en caso de serlo, si mejora el criterio de aspiración.

Es bueno señalar aquí que no todos los caminos están disponibles en cada iteración, sino que se trabaja con el método de generación de columnas. Para la generación de caminos se utilizan tres diferentes longitudes de arco, las cuales tiene en cuenta tanto los costos fijos como variables, así como la capacidad de la arista.

La búsqueda local continúa mientras no se alcance un número predefinido de generación de caminos para cada producto.

4.3 Fase de diversificación

El objetivo de esta fase es sacar la búsqueda del aparente óptimo local hacia una región prometedora. Para ello se define la vecindad discreta, en relación a las variables de diseño y se usa para modificar drásticamente la configuración de la red y diversificar la búsqueda.

La estrategia de diversificación implementada está basada en la observación que un número de "buenos" arcos aparecen una y otra vez en los caminos usados para satisfacer la demanda.

Por lo tanto se implementa una estructura de memoria de largo plazo basada en frecuencia, que registra por cuantas iteraciones ha estado un arco en la base, esto es por cuanto tiempo pertenece al menos a un camino básico.

Arcos que tengan puntaje alto en esta memoria usualmente han sido utilizados en las soluciones ya exploradas anteriormente.

Por ello para diversificar, uno selecciona un número pequeño de estos arcos y los quita de la base cualquier camino que contenga al menos uno de los arcos cerrados. Durante la tenencia tabú de estos arcos, ningún camino que los contenga podrá entrar a la base, a menos que el criterio de aspiración ignore el estatus tabú.

5 Resultados

Para evaluar el desempeño del algoritmo propuesto se generaron aleatoriamente 120 problemas, agrupados en las siguientes 4 clases:

Clase I: Costos Fijos altos, holgada en capacidad

Clase II: Costos fijos altos, apretada en capacidad

Clase III: Costos variables altos, holgada en capacidad

Clase IV: Costos variables altos, apretada en capacidad

Se consideraron también redes de diferentes tamaños: 30 nodos y 350 aristas (700 arcos), 50 nodos y 975 aristas (1950 arcos). Para cada tamaño se generaron instancias con 10, 50 y 100 productos. Todos ellos fueron corridos con Cplex 7.1 [4] durante 6 horas y con el algoritmo propuesto en este trabajo.

La tabla 1 muestra los resultados agrupando las instancias según las clases anteriormente definidas. La columna 1 muestra esta clasificación, la columna 2 muestra el promedio de las diferencias relativas entre la solución entregada por el algoritmo propuesto y la mejor solución entera entregada por el optimizador Cplex luego de 6 horas. Los valores negativos indican una mejora a favor del algoritmo desarrollado en este trabajo. En la tercera columna se ofrece el tiempo (medido en segundos) que necesitó el algoritmo propuesto.

Clase	TS vs Cplex	Tiempo-TS
I	−2.59%	72.37
II	5.33%	69.30
III	−2.23%	75.81
IV	−2.30%	66.05
General	−0.44%	70.88

Tabla 1: *Instancias agrupadas por clase*

Como era de esperar las instancias con costos fijos predominantes y capacidades muy justas (redes restringidas) obtuvieron los peores resultados.

En la tabla 2 se agrupan las instancias según el tamaño de la red. La primera columna muestra el número de nodos y productos de la red, la segunda columna muestra el promedio de la diferencia relativa del resultado obtenido por TS comparado contra la mejor solución entera entregada por Cplex. Los guiones indican que no pudo realizarse la comparación porque el optimizador no encontró ninguna solución factible en el período de 6 horas y al igual que en la tabla anterior los valores con signo negativo en la segunda columna indican una mejora a favor de la metaheurística empleada en este trabajo.

Nd-prod	Ts vs Cplex	Tiempo-Ts
30–10	1.35%	1.97
30–50	−1.85%	58.85
30–100	6.88%	88.90
50–10	−2.97%	27.40
50–50	-	62.10
50–100	-	117.80

Tabla 2: *Instancias agrupadas según sus dimensiones.*

Como puede observarse, para redes pequeñas el optimizador obtiene buenos resultados, sin embargo, a medida que crece el número de nodos y productos circulando en la red se justifica plenamente el uso de la metaheurística.

Por último, se realiza una comparación con los resultados entregados por un algoritmo basado en Scatter Search . En la tabla 3 se presentan los resultados para las redes de tamaño grande, donde se había detectado que se degradaba el desempeño de este último [1]. Al igual que en las tablas anteriores los signos negativos indican una mejoría contra lo que se está comparando.

Nodos-Productos	Ts vs SS	t-TS	t-SS
50–10	−3.93%	7.76	27.4
50–50	−5.71%	78.1	62.1
50–100	−11.03%	254.455	117.8

Tabla 3: *Comparación contra ScatterSearch*

Es bueno señalar que para redes más pequeñas el comportamiento de ambas metaheurísticas fue similar.

6 Conclusiones

En este trabajo se presenta un algoritmo que combina la metaheurística Búsqueda Tabú con el método Simplex Revisado para producir una búsqueda que explore el espacio de soluciones (variables de flujo) mediante movimientos que consisten en pivoteos del simplex revisado. Consideramos que el procedimiento presentado propone una forma eficiente de encontrar buenas soluciones factibles al problema de diseño de red multiproducto y constituye una buena adaptación, al caso no orientado que aquí se estudia, de desarrollos realizados para otro tipo de redes.

Por otra parte, consideramos que es posible extender y mejorar este trabajo para incorporar al TS otras técnicas tales como oscilación estratégica que permitan obtener resultados aún mejores, sobretodo en redes muy restringidas y con costos fijos predominantes, donde la inclusión o no de una arista puede hacer gran diferencia.

7 Bibliografía

[1] Ada M. lvarez, José Luis González-Velarde y Karim De-Alba. Grasp Embedded Scatter Search for the Multicommodity Capacitated Network Design Problem. Journal of Heuristics. Vol 11, Issue 3. pp 233-257, May 2005.

[2] lvarez, A; González-Velarde, J.L. y De Alba, K. (2001). "Un Algoritmo de Búsqueda para un Problema de Red capacitada multiproducto", Memorias del 3er Encuentro Internacional de Computación. Aguascalientes, Ags, 2001.

[3] B. Gendron, T.G. Crainic. Relaxations for Multicommodity Capacitated Network Design Problems, publication CRT-965, Centre de recherche sur les transports, Université de Montréal, 1994.

[4] B. Gendron, T.G. Crainic. Bounding Procedures for Multicommodity Capacitated Fixed Charge Network Design Problems, publication CRT-96- 06, Centre de recherche sur les transports, Université de Montréal, 1996.

[5] CPLEX Optimimization, Inc., Incline Village, NV. ILOG CPLEX 7.1 Reference Manual, 1999.

[6] F. Glover, M. Laguna. Tabu Search, Ed Klumer Academic Publisher, 1997.

[7] F. Glover. Tabu Search - Part I, ORSA Journal on Computing 1, pp 190-206, 1989.

[8] J.W. Herrmann, G. Ioannou, I. Minis, J.M. Proth. A Dual Ascent Approach to the Fixed-Charge Capacitated Network Design Problem, European Journal of Operational Research 95, pp 476-490. 1989.

[9] T.G. Crainic, M. Gendreau,, J. Farvolden. A Simplex-Based Tabu Search Method for Capacitated Network Design. INFORMS Journal on Computing, Vol. 12, No. 3, pp 223-236. 2000.

[10] T. Magnanti, R. Wong. Network Design and Transportation Planning: Models and Algorithms. Transportation Science 18, pp 1–55. 1984.