Revista Electrónica de Comunicaciones y Trabajos de ASEPUMA: Rect@

Resultados de diferentes experiencias con búsqueda local aplicadas a problemas de rutas

J.A. Pacheco y C.R. Delgado*

Universidad de BURGOS. Dpto. Economía Aplicada. Fac. CC EE y EE.
C/Francisco Vitoria s/n BURGOS 09006
e-mail: jpacheco@ubu.es cdelgado@ubu.es

Resumen:

En los problemas de optimización combinatoria siempre han tenido gran importancia los Algoritmos de Búsqueda Local (o de Búsqueda Vecinal) dentro de las técnicas heurísticas, especialmente a partir de la aparición de muchas de las recientes técnicas Metaheurísticas basadas en buena parte en el uso de estos movimientos vecinales. En este trabajo se analizan algunas experiencias relacionadas con estos, aplicados al VRPTW Mixto (carga y descarga). La primera de ellas es la definición y comparación de diferentes tipos de vecindarios para elegir el más adecuado; la segunda es la determinación de qué solución se elige en cada movimiento; la tercera es el uso de una técnica estrategia Búsqueda Local Rápida que puede servir para reducir considerablemente el tiempo de computación en problemas de gran tamaño. Para estas experiencias se usan instancias simuladas.

Palabras clave.- ***Problemas de Rutas, Búsqueda Local, Búsqueda Local Rápida***

* Recibido: Septiembre 1999
Aceptado Enero 2000, después de una revisión

1 INTRODUCCIÓN

El Problema de Rutas de Vehículos con Ventanas de tiempo y con Carga y Descarga Simultánea o sencillamente VRPTW Mixto (Mixed VRPTW) con flota heterogénea puede ser descrito de la forma siguiente: considérese un conjunto de puntos {2, 3, ..., n1} donde hay que entregar unas determinadas cantidades de mercancía q(i), i=2,...,n1, en forma de *palés*, desde un origen 1; además considérese otro conjunto de puntos {n1+1,n1+2,...,n1+n2} donde hay que recoger otras cantidades de *palés* q(i), *i* = n1+1,..., n1+n2, y llevarlas al origen 1. Para cumplir estos requerimientos se dispone de una flota con diferentes tipos de vehículos; cada tipo de vehículo tiene una capacidades de carga diferente; cada punto del problema lleva asociado un intervalo de tiempo de visita $[e_i, l_i]$, *i* = 2,..., n1+n2, (si se llega a i antes del instante e_i hay espera, y no puede visitar más tarde del instante l_i). Las distancias d_{ij} y tiempos t_{ij} entre cada par de puntos *i*, *j* $\in$ {1,2,...,n1+n2} son conocidas. (A partir de ahora *n* = n1+n2). El número de tipos de vehículos disponibles se denota por *ntipos* y las capacidades y costes de cada tipo por *capactipo(i)* y *costetipo(i)* , *i* = 1,..., *ntipos*; obviamente a más capacidad mas coste. No se consideran limitaciones en cuanto al número de vehículos disponibles por tipo ni en el total.

En este problema se ha de diseñar un conjunto de rutas de coste mínimo verificando las siguientes restricciones:

- Cada ruta comience y finalice en el punto 1;
- Se lleve la mercancía correspondiente a cada uno de los puntos del conjunto {2,..,n1}, y se recoja de cada uno de los puntos del conjunto {n1+1,n1+2,...,n1+n2}, (no es necesario que los puntos de descarga sean anteriores a los de recogida ni al revés) ;
- El número de *palés* que en cada momento lleva cada vehículo no supere su capacidad;
- Cada punto *i*, *i* = 2,.., n1+n2, sea visitado exactamente una vez:
- Se respeten los intervalos de tiempo de visita.

El coste total a minimizar *f* se descompone en dos partes:

- Una parte proporcional a la distancia total recorrida.
- El coste fijo por cada vehículo (según su tipo).

Este modelo es una generalización de dos subproblemas muy estudiados a lo largo de los años, el VRP con número libre de vehículos (n2 = 0, *ntipos* = 1, e_i = -inf., l_i= +inf. para *i* = 1,.., n1), y el VRPTW n2=0, *ntipos* = 1).

Existen muchos algoritmos de solución para el VRP y el VRPTW en la literatura. Se pueden encontrar recopilaciones de los principales en trabajos como los de Bodin y Golden (1.981), Desrochers y otros (1.988), Haouri y otros (1.990), Laporte (1.992) y Laporte y Osman, (1.995). En los últimos años ha tomado

importancia el desarrollo de algoritmos basados en procesos denominados Metaheurísticos como los *Algoritmos Genéticos, Temple Simulado, Búsqueda Tabú, GRASP, Búsqueda Local Guiada (GLS), Colonias de Hórmigas...* especialmente a partir de los trabajos de Gendreau y otros (1.991) ((1.994), en versión posterior), Osman (1.993) y más recientemente en los de Potvin y otros (1.993), (1.994), Thangiah y otros (1.993) y (1.994), Campos y Mota (1.995), Kantoravdis (1.995), Rochat y otros (1.995), Kilby y otros (1.997), Backer y otros, (1.997), Bullnheimer y otros (1.997) o Rego (1.998). Estos trabajos, en cualquier caso no considera el problema mixto objeto de este trabajo.

En este trabajo se analizan algunos aspectos relacionados con los procedimientos de Búsqueda Local o Búsqueda Vecinal para el VRPTW Mixto, que básicamente actúan de la forma siguiente:

Procedimiento Búsqueda Local
Leer una solución inicial
Repetir
Cambiar la solución actual por una solución en su vecindad que sea mejor mientras esta se encuentre

Esta estrategia siempre descendente (en el caso de minimizar) tiene dos grandes limitaciones: la convergencia a óptimos locales que no son globales en la mayoría de los casos y la dependencia de la solución inicial. Por tanto se tiende a usarla cada vez menos de forma aislada y cada vez más integrada en otras estrategias.

De hecho los procedimientos de Búsqueda Local están presentes en muchas estrategias usadas actualmente: *Búsqueda Local Guiada*, (Vondouris y Tsang (1.999)), *GRASP*, (Feo y Resende (1.989) y (1.995)) y *Concentración Heurística* (Fase I), (Rosing y Revelle (1.997)). Estas estrategias son en realidad repeticiones de procedimientos de Búsqueda Local (donde se modifica la función objetivo, la solución inicial, etc..); los *Algoritmos Meméticos* (Moscató (1.990)) añaden a las operaciones de los *Algoritmos Genéticos* la Búsqueda Local; también aparece en algunas fases de intensificación en *Búsqueda Tabú* (Glover (1.989) y (1.990)) y en *Búsqueda Dispersa* (Laguna (1.999)); la *Búsqueda Tabú* básica o el *Temple Simulado* (Kirkpatrick y otros (1.982) y (1.983)) son en realidad cadenas de movimientos vecinales.

Por tanto, un aspecto fundamental en el diseño de estas Metaestrategias, es el estudio y definición de los propios vecindarios y los procedimientos de búsqueda local que están integrados en dichas Metaestrategias. Concretamente en este trabajo se estudian, para el modelo que se considera, los siguientes aspectos: la incidencia de diferentes tipos de vecindario así como la de diferentes criterios de elección de nueva solución. Además, como aspecto a nuestro entender más relevante, se

analiza el efecto de ciertas modificaciones, *Búsqueda Local Rápida*, para ahorrar tiempo de computación.

En las tres siguientes secciones, se describen las diferentes definiciones de *vecindarios* usados. En la quinta sección se muestra el seudocódigo de los algoritmos empleados. En la sexta se muestran y comparan los resultados obtenidos con los diferentes vecindarios. En la séptima se compara el uso de diferentes criterios para la elección de una nueva solución. En la octava y novena se analiza el efecto de la incorporación a este modelo de una serie de modificaciones que dan lugar a lo que se denomina *Búsqueda Local Rápida*.

A partir de ahora se denotará por *S* el conjunto de soluciones factibles del problema, y *f* la función de costes a minimizar definida en *S*.

2 VECINDARIO TIPO OR

Se van a considerar tres tipos de soluciones vecinas, uno basado en la idea propuesta por Or (1.976), para el *Problema del Viajante* o TSP; otro más reciente que extiende el usado en los trabajos de Gendreau et al (1.991), Osman (1.993), Campos y Mota (1.995) y Kilby y otros (1.997), que se explicará en la tercera sección; en la cuarta sección se describirá el vecindario propuesto por Taillard y otros (1.997).

Se van a definir como soluciones vecinas las obtenidas por el método de intercambio propuesto por Or (1.976) que sean factibles. El método de intercambio de Or es una variante de los conocidos intercambios r-óptimos desarrollados por Lin (1.965) y Lin & Kernighan (1.973) para el TSP simétrico. Como se verá más adelante, el método de Or se puede utilizar en problemas asimétricos. La eficacia de este método para el TSP ha sido contrastada en trabajos como el de Nurmi (1.991).

Obsérvese que una solución puede expresarse de forma sencilla, como una única secuencia de puntos; por ejemplo, la solución formada por las dos rutas siguientes

ruta 1: 1 - 3 - 5 - 1; *ruta 2*: 1 - 4 - 2 - 1

puede expresarse como:

1 - 3 - 5 - 1 - 4 - 2 - 1

los '1' en posiciones intermedias representan la vuelta de un vehículo al origen y la salida del siguiente (obviamente el último y el primer elemento siempre serán '1').

De esta forma, como se ilustra a continuación, se puede aplicar los Or-intercambios para obtener soluciones vecinas de la actual, considerando a esta como una única secuencia.

Or propone restringir la búsqueda de intercambios a los 3-intercambios en los que cadenas[1] de uno, dos o tres puntos consecutivos son recolocadas entre otros dos. Nótese que con estos intercambios no se cambia el sentido de los diferentes tramos.

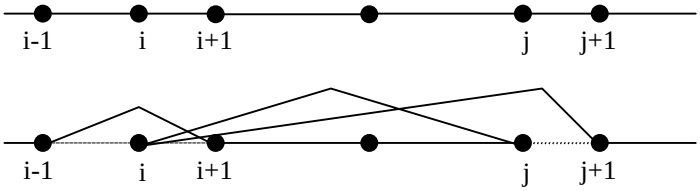

Figura 1.- Posible recolocación del elemento i entre j y j+1.

En nuestro caso seguiremos la misma idea, pero sólo consideraremos recolocaciones hacia adelante y no pondremos límite al tamaño de la cadena a recolocar (salvo el determinado por el tamaño del problema); además, se debe comprobar la factibilidad de cada posible recolocación respecto de las restricciones del problema. A continuación se ilustra la recolocación de una cadena de *k* elementos comenzando en *i* entre *j* y *j*+1.

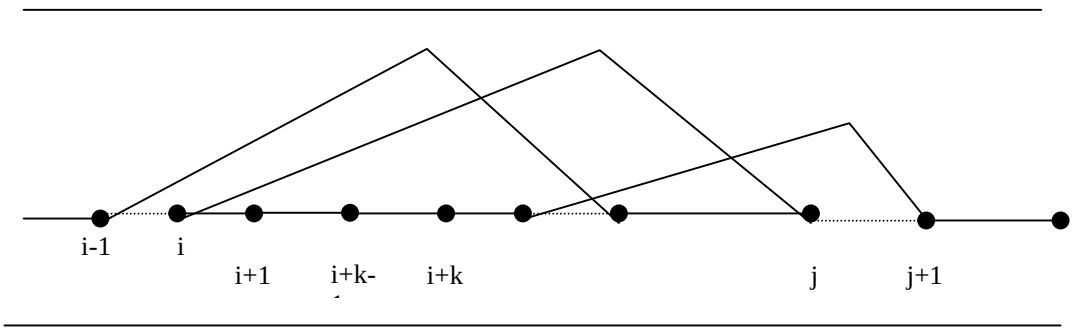

Figura 2.- Recolocación de una cadena de k elementos.

[1]En este trabajo se denomina *cadena* a toda secuencia de puntos consecutivos en la solución actual.

Para cada $s \in S$ denotaremos por $N^k_1(s)$ como el conjunto de soluciones factibles obtenidas por recolocaciones hacia adelante de cadenas de a lo sumo *k* elementos en *s*. Se denota por $N^{\infty}_1(s)$ el conjunto de soluciones factibles obtenidas por todas las recolocaciones.

Sea *m* el número de rutas que componen una solución $s \in S$, el número de puntos que contiene dicha solución, considerada como una única cadena, es de n+m. Como cada intercambio viene determinado por la posicion de los puntos *i* y *j* y el tamaño *k* de la cadena a recolocar, se tiene que para *k* finito el número de intercambios a considerar para obtener $N^k_1(s)$ es de $O(n^2)$, y para $N^{\infty}_1(s)$ es de $O(n^3)$.

3 VECINDARIOS TIPO GENDREAU-CLARKE

El segundo tipo de vecindarios no considera las soluciones como una única secuencia de puntos, sino que sólo considera 3 tipos de intercambios entre 2 rutas diferentes:

- Tipo I: Intercambio del elemento *i* de la ruta *r* con el elemento *i'* de la ruta *r'* :

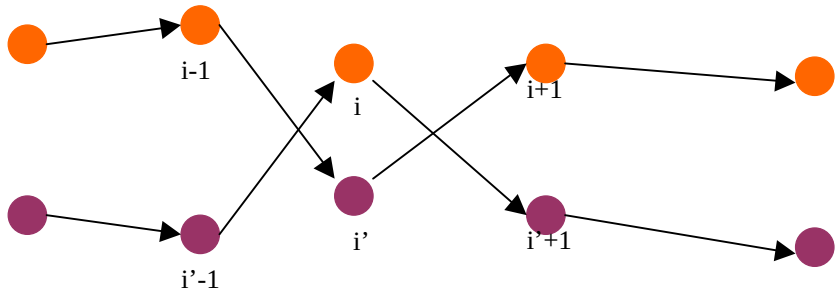

Figura 3.- Eliminación de los arcos (i-1,i), (i,i+1), (i'-1,i'), (i' i'+1) e incorporación de los arcos (i-1,i'), (i',i+1), (i'-1,i), (i, i'+1).

- Tipo II: Inserción del elemento i de la ruta r entre los elementos i' e i'+1 de la ruta r':

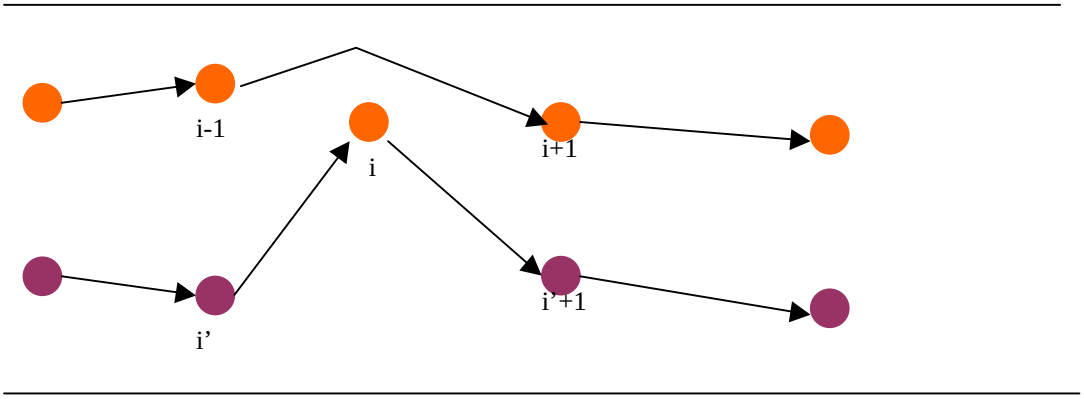

Figura 4.- Eliminación de los arcos (i-1,i), (i, i+1), (i',i'+1), e incorporación de los arcos (i',i), (i,i'+1), (i-1,i+1)

Tipo III: Cruce de las rutas de la ruta *r* y *r*' por los elementos *i* e *i*' según la figura 5

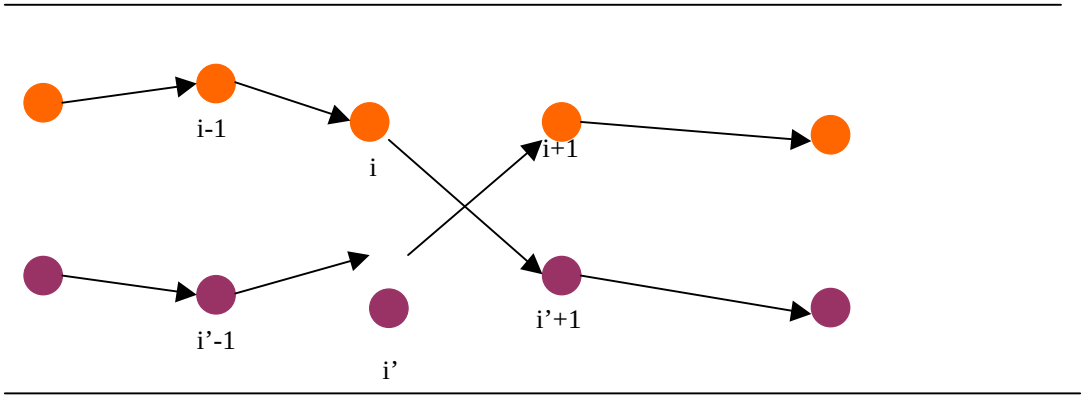

Figura 5.- Eliminación de los arcos (i,i+1), (i' i'+1) e incorporación de los arcos (i',i+1), (i, i'+1).

Los dos primeros tipos aparecen en el trabajo de Gendreau y otros (1.991); el tercero aparece en el trabajo de Kilby y otros (1.997), y se basa en algunas de las ideas propuestas por Clarke and Wright, (1.964).

Obsérvese que el tipo I es en realidad un 4-intercambio, el tipo II un 3-intercambio y el tipo III un 2-intercambio. Para cada $s \in S$ se denotará a los conjuntos de soluciones factibles generadas por cada uno de estos 3 tipos de intercambio por $N^1_2(s)$, $N^2_2(s)$ y $N^3_2(s)$ respectivamente; así mismo se denota por $N_2(s)$ a la unión de estos 3 conjuntos.

Obsérvese que, en realidad, cada intercambio de este apartado viene determinado por los puntos *i* e *i'* de 2 rutas diferentes. En una solución *s* con m rutas hay que considerar $\theta(m^2)$ pares de rutas, y en cada ruta aproximadamente n/m elementos; por tanto el número de intercambios a considerar para obtener $N_2(s)$ es de $\theta(m^2 \cdot (n/m) \cdot (n/m)) = \theta(n^2)$.

4.- VECINDARIOS TIPO TAILLARD

El tercer tipo de vecindario, propuesto por Taillard y otros (1.997), considera intercambios de cadenas de puntos (de tamaños *k* y *k*1) entre dos rutas diferentes, (CROSS intercambios) según muestra la figura 6.

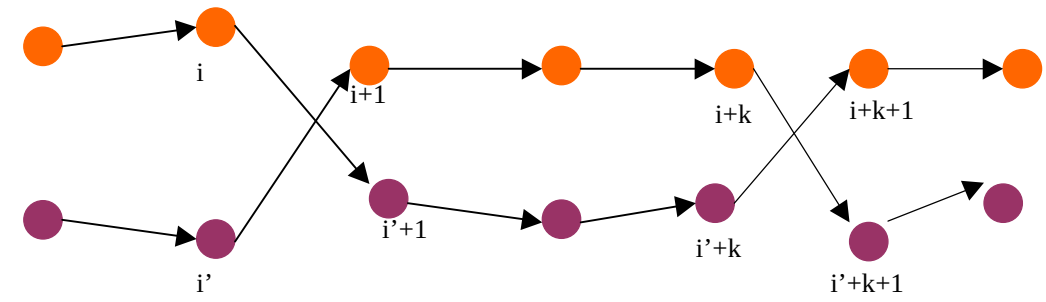

Figura 6.- Eliminación de los arcos (i, i+1), (i+k, i+k+1), (i1, i1+1), (i1+k1, i1+k1+1) e incorporación de los arcos (i, i1+1), (i+k, i1+k1+1), (i1, i+1), (i1+k1, i+k+1).

Obsérvese que estos intercambios generalizan los descritos en el apartado anterior:

- si $k = k1 = 1$, se tiene un intercambio tipo I;
- si $k = 1$ y $k1 = 0$ intercambio tipo II;
- si $i+k+1$ e $i1+k1+1$ coinciden con el final de la ruta (punto 1) intercambio tipo III.

Para cada $s \in S$ se denotará por $N^k_3(s)$ a los conjuntos de soluciones factibles generadas por estos CROSS intercambios de cadenas de a lo sumo *k* elementos;

análogamente se denota por $N^{\infty}_3(s)$ el conjunto de soluciones factibles generadas por todos estos CROSS intercambios.

Obsérvese que cada intercambio de este apartado viene determinado por las 2 cadenas que se intercambian pertenecientes a rutas diferentes r y r'; cada cadena a su vez viene determinada por sus elementos inicial y final. Por tanto, en una solución *s* con m rutas hay que considerar $\theta(m^2)$ pares de rutas y en cada ruta aproximadamente n/m elementos, por consiguiente $\theta(n^2/m^2)$ cadenas. De esta forma el número de intercambios a considerar para obtener $N_3(s)$ es de $\theta(m^2 \cdot (n^2/m^2) \cdot (n^2/m^2)) = \theta(n^4/m^2)$.

El chequeo de la factibilidad y la valoración de cada intercambio (tanto de tipo Or como de tipo Gendreau o Taillard) puede suponer un tiempo de computación excesivo. En el trabajo de Pacheco y Delgado (1.997) se propone el uso de variables globales, con las que el número de operaciones para chequear y evaluar cada intercambio es constante, es decir, independiente del tamaño del problema.

5 DISEÑO DE LOS ALGORITMOS A USAR

Como se ha comentado en la introducción se van a incorporar estos vecindarios en procedimientos de Búsqueda Local para su comparación. A continuación se muestran en seudocódigo los siguientes procedimientos:

Procedimiento Movimiento_Vecinal_Or(*k, sf*)
Determinar $s' \in N^k_1(sf)$ *verificando* $f(s') = \min\{f(s) / s \in N^k_1(sf)\}$
Si $f(s') < f(sf)$ *entonces hacer* $sf = s'$

Procedimiento Búsqueda_Local_Or(*k, sf*)
Repetir
Ejecutar Movimiento_Vecinal_Or(*k, sf*)
hasta que $f(s) \geq f(sf), \forall s \in N^k_1(sf)$.

Análogamente:

Procedimiento Movimiento_Vecinal_Ge(*k, sf*)
Determinar $s' \in N_2(s)$ *verificando* $f(s') = \min\{f(s) / s \in N_2(sf)\}$
Si $f(s') < f(sf)$: *hacer* $sf = s'$
Ejecutar Búsqueda_Local_Or(*k, r'*) y Búsqueda_Local_Or(*k, r'*) *donde r' y r'' son las dos rutas de sf modificadas para dar lugar a s'*

Procedimiento Búsqueda_Local_Ge(*k*, *sf*)
Repetir
Ejecutar Movimiento_Vecinal_Ge(*k*, *sf*)
hasta que f(s) ³ *f(sf), " s* Î $N^k_1(sf)$

Y finalmente:

Procedimiento Movimiento_Vecinal_Ta(*k*, *sf*)
Determinar s' Î $N_2(s)$ *verificando f(s') = min{f(s) / s* Î $N^{\infty}_3(sf)$*}*
Si f(s') < f(sf) : *hacer sf = s'*
Ejecutar Búsqueda_Local_Or(*k*, *r'*) y Búsqueda_Local_Or(*k*, *r'*) *donde r' y r'' son las dos rutas de sf modificadas para dar lugar a s'*

Procedimiento Búsqueda_Local_Ta(*k*, *sf*)
Repetir
Ejecutar Movimiento_Vecinal_Ta(*k*, *sf*)
hasta que f(s) ³ *f(sf), " s* Î $N^{\infty}_3(sf)$

Obsérvese como en Búsqueda_Local_Ge(k, sf) y *Búsqueda_Local_Ta*(*k*, *sf*) cada movimiento vecinal se compone de un intercambio de cadenas entre dos rutas diferentes y de la mejora de cada una esas dos rutas, siguiendo la idea de Osman (1.993).

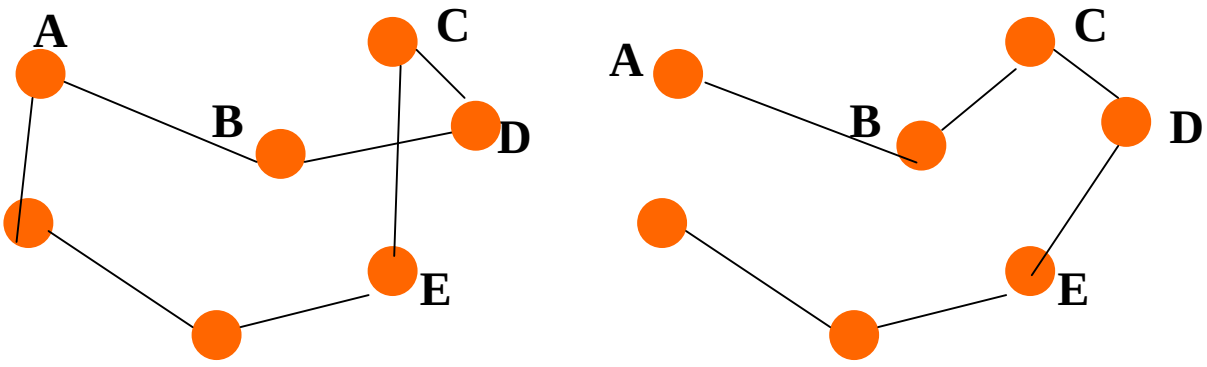

Figura 7.- Tras la recolocación de la cadena formada por los elementos C - D entre los elementos A y B, la ruta resultante puede ser mejorada posteriormente

6 COMPARACIÓN DE LOS PROCEDIMIENTOS DE BÚSQUEDA LOCAL CON DIFERENTES VECINDARIOS CON INSTANCIAS SIMULADAS

Se van a comparar los procedimientos *Búsqueda_Local_Or* con $k = 3$ y $k = \infty$ y *Búsqueda_Local_Ge* y *Búsqueda_Local_Ta* con k = 3. Para ello se han considerado dos tipos de instancias, el primero con descarga pura (n2 = 0), y el segundo con igual número de clientes de carga y descarga (n1 = n2); para cada tipo se consideran 10 tamaños deferentes: 10 puntos, 20, ... hasta 100 puntos. Para cada tipo y tamaño se generan 10 instancias. Los datos de cada instancia se definen de la forma siguiente:

- Se asigna a cada punto del problema dos coordenadas *x* e *y*, cuyos valores son generados aleatoriamente con distribución uniforme entre 0 y 100. La distancia entre cada par de puntos se define como la distancia euclídea correspondiente. Los tiempos (en minutos) de trayecto se toman igual a la distancia (en Kms.). (Es decir consideramos una velocidad de 60 kms/hora). El coste por Km. es de 1 u.m.
- A e_i y l_i, para $i = 2,...,$ n1+n2, se les asigna respectivamente dos valores enteros aleatorios generados uniformemente: entre 600 y 720 (minutos) en el primer caso y entre 960 y 1.080 en el segundo. Se hace e_1 igual a 480 y l_1 igual a 1.200.
- A cada q(i), $i = 2, ...,$ n1+n2, se le asigna un valor entero generado uniformemente de forma aleatoria entre 1 y 12.
- En todos los casos se supone dos tipos de vehículos, con capacidades 10 y 20, y con costes 1.000 y 1.200 u.m. respectivamente.

Los 4 procedimientos de Búsqueda local utilizan la misma solución inicial obtenida por una adaptación a este modelo de un algoritmo de *inserción* que se describe en el apéndice. En las tablas siguientes se muestran con los resultados medios de los costes totales (costes por distancia y por tipo de vehículo según lo descrito en el párrafo anterior) y los tiempos de computación, en segundos, según el tipo de instancia y tamaño. Todas los algoritmos descritos y programas usados para la realización de este trabajo han sido programados usando los compiladores BORLAND PASCAL (ver.7.0) y BORLAND DELPHI (3.0). El equipo usado es un Ordenador Personal PENTIUM MMX 200 mhz. Quedamos a disposición de los lectores interesados en dichos programas para la verificación de los resultados.

Tipo I (n2 = 0)

Tamaño	**Solución Inicial**	**B. Local_Or (k=3)**	**B. Local_Or (k=∞)**	**B. Local_Ge (k=3)**	**B. Local_Ta (k=3)**
10	4240.4 0.0091	4076.4 0.0020	4058.3 0.0070	4029.0 0.0150	4021.4 0.0130
20	7786.6 0.0221	7629.8 0.0180	7498.9 0.0750	7486.1 0.0621	7459.5 0.0992
30	12126.4 0.0602	11815.5 0.0390	11768.1 0.3476	11707.1 0.2653	11673.9 0.2653
40	16098.6 0.1222	15594.8 0.1341	15292.9 1.1598	15264.1 0.3678	15355.7 0.7179
50	19631.0 0.2143	19215.7 0.1523	18903.3 2.1961	18796.8 0.6009	18659.8 1.2498
60	23887.1 0.3504	23317.3 0.3115	22775.3 4.8688	22577.7 1.0233	22550.4 2.1552
70	27314.1 0.5378	26762.0 0.4026	26112.1 9.5955	25937.7 1.5423	25951.7 3.7375
80	31556.0 0.7650	31047.1 0.4387	30234.2 18.1501	29945.0 2.2281	29830.8 5.2667
90	34568.7 1.0727	33994.5 0.8281	33114.2 27.2734	32984.1 3.6935	32937.7 8.0858
100	39481.6 1.4089	39053.6 0.8914	37989.3 50.5523	37770.0 4.2954	37681.8 10.6062

Tipo II (n1 = n2)

Tamaño	**Sol.Inicial**	**B.Local_Or (k=3)**	**B.Local_Or (k=∞)**	**B.Local_Ge (k=3)**	**B.Local_Ta (k=3)**
10	2860.5 0.0090	2793.9 0.0070	2787.1 0.0050	2779.4 0.0110	2778.2 0.0120
20	4890.8 0.0330	4585.9 0.0210	4579.2 0.0511	4544.0 0.0472	4523.9 0.0902
30	6865.7 0.0911	6670.7 0.0411	6295.2 0.2354	6377.0 0.1043	6360.7 0.2783
40	9417.5 0.1927	9292.0 0.0900	9041.8 0.7401	9054.0 0.2124	8869.9 0.6569
50	11488.1 0.3674	11088.4 0.1711	10755.7 1.4770	10914.5 0.4226	10720.1 1.5293
60	14183.7 0.5887	13515.6 0.4167	13187.0 4.0559	13222.0 0.8834	13220.7 3.2409
70	15727.7 0.9465	15360.0 0.4897	14737.0 6.5393	14603.4 1.3748	14725.4 5.2313
80	17424.8 1.5972	17043.1 0.6250	16647.6 11.3062	16611.3 1.9637	16369.4 8.0787
90	20075.4 1.9460	19388.7 0.9323	18703.9 17.0927	18696.2 2.8691	18598.5 11.2082
100	21790.9 2.6690	21092.3 1.2338	20641.5 23.6641	20850.7 3.4097	20661.7 14.4868

Las conclusiones en este apartado son claras:

- Los procedimientos basados en intercambios de Or dan lugar a peores soluciones que los basados en intercambios tipo Gendreau y su generalización (los intercambios tipo Taillard o CROSS intercambios). Solamente tomando k = ∞, y con instancias del segundo tipo, los intercambios de Or parecen poder "competir" con los otros dos tipos, pero siempre a costa de utilizar un tiempo de computación mayor.
- Por otra parte los CROSS intercambios de Taillard consiguen, por lo general, mejores soluciones que los de tipo Gendreau y, en definitiva, las mejores soluciones de todos. En cuanto al tiempo de computación los CROSS intercambios emplean más que los de Gendreu, pero sustancialmente menos que los de Or, para k = ∞.

7 COMPARACIÓN DE DIFERENTES CRITERIOS DE ELECION DE NUEVAS SOLUCIONES

En la sección anterior, en todos los casos, los procedimientos de búsqueda local cambiaban en cada paso a la mejor solución vecina (*Mayor Descenso*). Sin embargo no es el único criterio existente. Otras opciones, por ejemplo, son: elegir aleatoriamente una solución que mejore la actual (*descenso aleatorio*), o elegir la primera que mejora en la exploración del vecindario (*primer descenso*).

Algunas experiencias, como en Laguna y otros (1.994) y (1.998), indican que el criterio de *Mayor Descenso* no necesariamente lleva a mejores soluciones finales, ya que en muchos casos puede llevar al proceso a "encajonarse" prematuramente en mínimos locales muy cercanos a la solución inicial. La estrategia de *primer descenso*, ahorra tiempo de computación al no explorar todo el vecindario, y puede conducir a mejores soluciones.

Para analizar este punto en nuestro problema, a continuación se van a comparar los dos criterios *Mayor Descenso* y *Primer Descenso* en procedimientos de Búsqueda Local con vecindarios tipo Taillard. El criterio de mayor descenso fue definido y usado en la sección anterior; para obtener el procedimiento de Búsqueda Local con *primer descenso* sustituimos en *Movimiento_Vecinal_Ta*(*k, sf*):

$$\textit{Determinar } s' \in N_2(s) \textit{ verificando } f(s') = \min\{f(s) / s \in N^{\infty}_3(sf)\}$$

por

$$\textit{Tomar } s' \textit{ el primer elemento de } N_2(s) \textit{ verificando } f(s') < f(sf).$$

Para comparar ambos criterios se han generado el mismo tipo de instancias, tamaños, número, etc que en la sección anterior. La solución inicial también se ha generado como en la sección anterior. A continuación se muestran los resultados medios (costes y tiempo de computación) por tamaño.

Tipo I (n2 = 0)

Tamaño	B.Local_Ta (k=3) Mayor Descenso		B.Local_Ta (k=3) Primer Descenso	
	Coste	**Tpo. Comput.**	**Coste**	**Tpo. Comput.**
10	3713.2	0.0220	3723.1	0.0231
20	7526.9	0.0881	7538.5	0.0781
30	11516.3	0.2965	11428.0	0.2483
40	15554.0	0.5607	15554.1	0.4917
50	19555.5	1.2518	19614.9	0.8771
60	22327.6	2.0931	22490.9	1.3169
70	26548.2	3.6424	26453.4	2.2282
80	31048.6	5.7562	31016.5	3.3250
90	32708.5	7.6140	32844.7	4.4865
100	36613.2	11.3324	36738.3	6.2239

Tipo II (n2 = n1)

Tamaño	B.Local_Ta (k=3) Mayor Descenso		B.Local_Ta (k=3) Primer Descenso	
	Coste	**Tpo. Comput.**	**Coste**	**Tpo. Comput.**
10	2286.2	0.0150	2289.0	0.0110
20	4669.4	0.0901	4562.1	0.0882
30	6594.4	0.2652	6592.8	0.2383
40	8593.5	0.6408	8795.3	0.4897

Tamaño	B.Local_Ta (k=3) Mayor Descenso		B.Local_Ta (k=3) Primer Descenso	
	Coste	**Tpo. Comput.**	**Coste**	**Tpo. Comput.**
50	10380.3	1.6875	10421.7	1.0366
60	12714.4	2.5838	12789.4	1.5663
70	14226.7	4.6617	14343.5	2.8724
80	16877.7	7.6078	17015.6	4.2925
90	18554.3	10.3379	18536.8	6.0424
100	20839.4	16.5227	20782.4	10.0444

Los resultados indican que el criterio de *Mayor descenso*, lleva por lo general y en término medio a soluciones ligeramente mejores; pero en bastantes ocasiones, según se ha comentado anteriormente y como se puede apreciar en la tabla, es el criterio de *Primer Descenso* quien aporta las mejores soluciones y empleando además un tiempo de computación ligeramente inferior.

8 EFECTOS DEL USO DE BÚSQUEDA LOCAL RÁPIDA

En esta sección adaptamos a este modelo una idea propuesta por Bentley (1.992) para el TSP y usada por Vondouris y Tsang (1.995), para este mismo modelo y que puede ahorrar mucho tiempo de computación en procedimientos de Búsqueda Local sin perjuicio de la solución obtenida.

La idea consiste básicamente en dividir el vecindario de la solución actual en subvecindarios más pequeños y asociar a cada uno de estos subvecindarios una variable binaria que indique si están activos o inactivos, de forma que en cada paso sólamente se exploran los subvecindarios activos. Inicialmente todos los subvecindarios están activos. En cada iteración, si al explorar un subvecindario este no contiene alguna solución que mejore la actual pasa a ser inactivo, en caso contrario permanece activo. Por otra parte, si, como resultado de un movimiento, un subvecindario se modifica entonces pasa a ser activo.

De esta forma a medida que el proceso avanza y la solución mejora, habrá menor número de subvecindarios activos y por tanto se gastará menos tiempo de computación en exploraciones innecesarias sin que, obviamente, quede afectada la

solución final. Esta pequeña modificación da lugar a lo que se denomina *Búsqueda Local Rápida*.

En este sentido, cuanto mayor sea el número de subvecindarios que se consideren más tiempo de computación se consigue ahorrar como ilustra la figura siguiente: se representa con el círculo grande de la izquierda un subvecindario grande de la solución actual sin subdividir, y con el círculo grande de la derecha el mismo subvecindario dividido en subvecindarios más pequeños. Los círculos pequeños en blanco representan soluciones que no mejoran la actual y el círculo pequeño en negro una que sí mejora la actual. Esta solución obliga, en el primer caso, a explorar todo el subvecindario grande en los siguientes pasos ya que estaría activo; mientras que en el segundo caso sólo es necesario explorar el subvecindario pequeño que contiene la solución que mejora la actual ya que los demás estarían inactivos.

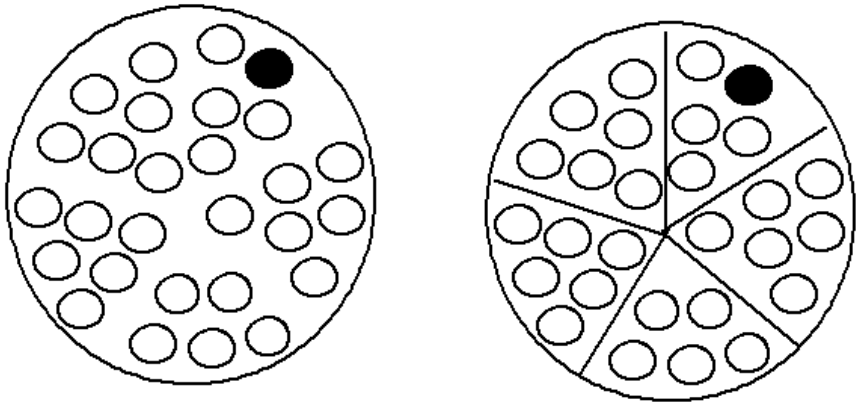

Figura 8.- Subvecindario sin dividir: hay que explorarlo todo. Subvecindario dividido: no hay que explorar las partes que no contengan mejora

En nuestro caso se usa la mima idea básica aplicada a *Búsqueda_Local_Ta*. El conjunto de intercambios tipo Taillard lo vamos a dividir en subconjuntos de la siguiente forma: para cada solución *s* y para cada par de puntos *i* e *i*1 pertenecientes a rutas diferentes (incluimos los 1 iniciales) se define:

$N_3(i, i1)$: Conjunto de intercambios entre cadenas, que tienen su origen en $i+1$, e $i1+1$ respectivamente. Cada cadena sólo puede contener elementos de una misma ruta y no puede contener al 1. El tamaño de alguna de las cadenas puede ser 0 (como se indicó en la sección 4).

Así se tiene que $N^{\infty}_3(s)$ se puede poner como unión disjunta de los subconjuntos $N_3(i, i1)$, para todos los pares de elementos *i* e *i*1 pertenecientes a rutas diferentes.

Fácilmente se puede guardar el carácter activo o inactivo de todos los subconjuntos en una matriz booleana, con gasto de memoria pequeño.

De esta forma cuando un subconjunto $N_3(i, i1)$ se ha explorado sin encontrarse mejora se debe desactivar; cuando se realiza un intercambio se deben hacer activos los subconjuntos $N_3(i, i1)$ donde o bien i o bien $i1$ sea alguno de los elementos que aparecen en alguna de las rutas afectadas por el intercambio.

Se va a analizar el efecto de incorporar esta idea al procedimiento *Búsqueda_Local_Ta*, tanto con el uso del criterio de *primer descenso* como con el de *mayor descenso*. Para ello se ha generado el mismo tipo de instancias, tamaños, número, solución inicial y forma de definir los datos que en la sección anterior. La solución inicial también se ha generado como en la sección anterior. A continuación se muestran los tiempos medios de computación por tamaño y tipo

Tipo I (n2 = 0)

	B.Local_Ta (k=3) Mayor Descenso		**B.Local_Ta (k=3) Primer Descenso**	
Tamaño	**Sin Búsqueda Local Rápida**	**Con Búsqueda Local Rápida**	**Sin Búsqueda Local Rápida**	**Con Búsqueda Local Rápida**
10	0.0231	0.0190	0.0200	0.0221
20	0.0910	0.0783	0.0960	0.0913
30	0.3026	0.1973	0.2393	0.1894
40	0.6087	0.3257	0.4527	0.3112
50	1.3278	0.5547	1.0245	0.5339
60	2.3944	0.8873	1.4821	0.7571
70	3.7572	1.1877	2.2452	1.0085
80	5.3517	1.4973	3.3076	1.3201
90	7.7070	1.9498	3.9675	1.5321
100	10.9106	2.4966	6.3632	2.2200

Tipo II (n2 = n1)

Tamaño	B.Local_Ta (k=3) Mayor Descenso		B.Local_Ta (k=3) Primer Descenso	
	Sin Búsqueda Local Rápida	**Con Búsqueda Local Rápida**	**Sin Búsqueda Local Rápida**	**Con Búsqueda Local Rápida**
10	0.0170	0.0090	0.0140	0.0110
20	0.0842	0.0792	0.0701	0.0721
30	0.3624	0.2966	0.2571	0.2365
40	0.7872	0.5088	0.5669	0.4306
50	1.5615	0.8572	1.1798	0.7513
60	2.9594	1.4069	1.8558	1.1328
70	4.0508	1.6563	2.4434	1.3200
80	6.7858	2.5828	3.8936	1.9608
90	9.8130	3.2554	5.6792	2.7121
100	15.4442	4.7399	6.7888	2.9441

A la vista de los resultados es claro el efecto positivo del uso de la Búsqueda Local Rápida: el tiempo de computación se reduce sustancialmente, y más a medida que el tamaño del problema aumenta. En el caso de las instancias de tamaño 100 el tiempo de computación llega a ser 5 veces menor.

Por otra parte, la reducción es ligeramente inferior, aunque sigue siendo muy significativa, (3 veces más rápido) con el criterio de *Primer Descenso*; la explicación es clara: al tomar la primera solución que mejora la actual no se exploran todos los intercambios, pudiendo quedar sin explorar subconjuntos activos, que de esta forma no se desactivan aunque no ofrezcan mejoras. En cualquier caso la reducción es importante. Finalmente, y relacionado con este último aspecto, los tiempos de computación de ambos criterios (Mayor Descenso y Primer Descenso), que eran en principio ligeramente diferentes, con el uso de Búsqueda Local Rápida tienden a igualarse mucho.

A continuación se muestra una gráfica de la evolución de los tiempos de computación para las instancias del primer tipo.

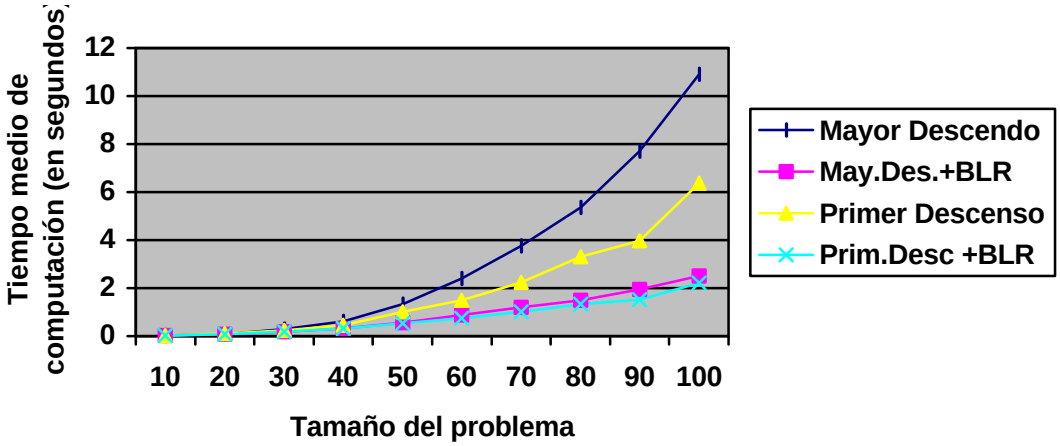

Figura 9.- Evolución de los tiempos medios de computación según el tamaño (instancias tipo I).

9 MODIFICACIÓN EN BÚSQUEDA LOCAL RÁPIDA

Por los resultados del apartado anterior es evidente el efecto positivo de las modificaciones que dan lugar a la Búsqueda Local Rápida: división del vecindario e incorporación del carácter activo o inactivo a cada subvecindario. De esta forma sólo se exploran los subvecindarios que se hayan modificado en la iteración anterior o que contengan mejoras en la solución actual.

Sin embargo, en este trabajo consideramos que se podría llegar un poco más lejos: ¿Para qué repetir la exploración de un vecindario que no se ha modificado aunque tenga mejoras? ¿No sería mejor usar la información encontrada en pasos anteriores sobre ese subvecindario en vez de repetir la exploración?

Para llevar a cabo esto, en este apartado se propone asociar a cada subvecindario una variable binaria que indique si se ha modificado o no en la última iteración y otra variable que contenga la información necesaria sobre la mejor solución de ese subvecindario. La forma de actuar es parecida a *Búsqueda Local Rápida*: inicialmente se exploran todos los subvecindarios, en las siguientes iteraciones sólo se exploran los subvecindarios modificados en la iteración anterior; cuando un subvecindario se explora se guarda la información sobre su mejor solución; de los subvecindarios no modificados sólo se considera su mejor solución sin que haga falta explorar el resto.

Así, es posible esperar mejoras muy ligeras en los tiempos de computación sobre *Búsqueda Local Rápida* al no tener que explorar subvecindarios no

modificados de pasos anteriores aunque contengan mejoras, ya que se guarda información sobre estas.

Se analiza el efecto de estas modificaciones en *Búsqueda_Local_Ta* como en la sección anterior. Más concretamente, para subconjuntos $N_3(i, i1)$ se asocia una variable booleana que indique si se ha modificado o no; y un registro que guarde el tamaño *k* y *k*1 de las cadenas que intervienen en el mejor intercambio de N_3(i, i1) y el valor de la ganancia (o pérdida) de dicho intercambio. Obsérvese que en este caso se usa mucha más memoria que en Búsqueda Local Rápida, ya que, además de usarse una matriz booleana, se usa una matriz de una variable compuesta por dos valores enteros (tamaños de las cadenas) y uno real (ganancia/pérdida).

Se compara la evolución de los tiempos de computación sin usar *Búsqueda Local Rápida*, con *Búsqueda Local Rápida* y *Búsqueda Local Rápida Modificada* (propuesta en esta sección); en todos los casos se usa el criterio de *mayor descenso*. Para ello se han simulado los mismos tipos de instancias, forma de definir los datos, y solución inicial que en anteriores secciones. El tamaño en este caso llega hasta 200 puntos y se han generado 10 instancias por cada tamaño. A continuación se muestran los tiempos medios de computación en segundos por tamaño y tipo.

Tipo I (n2 = 0)

	B. Local_Ta (k=3) Mayor Descenso		
Tamaño	**Sin Búsqueda Local Rápida**	**Búsqueda Local Rápida**	**Búsqueda Local Rápida Modificada**
10	0.0230	0.0210	0.0211
20	0.0790	0.0682	0.0712
30	0.2814	0.1785	0.1793
40	0.7181	0.3866	0.3897
50	1.2726	0.5597	0.5657
60	2.3384	0.8553	0.8502
70	3.8106	1.2278	1.2147
80	5.3839	1.5009	1.4691

Tamaño	B. Local_Ta (k=3) Mayor Descenso		
	Sin Búsqueda Local Rápida	**Búsqueda Local Rápida**	**Búsqueda Local Rápida Modificada**
90	8.4261	2.1079	2.0629
100	12.0320	2.7220	2.6280
110	16.5340	3.9037	3.7835
120	20.1770	4.3652	4.2289
130	28.0233	5.6142	5.3788
140	35.3428	6.5306	6.2449
150	44.2526	7.7021	7.3106
160	52.2871	8.3791	7.9394
170	63.7826	9.8651	9.2364
180	77.1929	11.3262	10.6244
190	89.5038	12.6432	11.8860
200	108.5680	14.4858	13.5004

Tipo II (n1 = n2)

Tamaño	B. Local_Ta (k=3) Mayor Descenso		
	Sin Búsqueda Local Rápida	**Búsqueda Local Rápida**	**Búsqueda Local Rápida Modificada**
10	0.0340	0.0321	0.0330
20	0.1091	0.1103	0.1182

Tamaño	B. Local_Ta (k=3) Mayor Descenso		
	Sin Búsqueda Local Rápida	**Búsqueda Local Rápida**	**Búsqueda Local Rápida Modificada**
30	0.4346	0.3636	0.3735
40	0.7752	0.5497	0.5610
50	1.6252	0.9656	0.9845
60	3.6713	1.8326	1.8595
70	5.1062	2.2893	2.3182
80	8.6466	3.3387	3.3756
90	13.4894	4.6957	4.7430
100	15.3832	4.9472	4.9851
110	22.6018	6.7879	6.8611
120	29.4053	7.9507	7.9868
130	42.8997	10.4511	10.5020
140	48.6099	11.2151	11.2302
150	67.6903	14.6719	14.5929
160	82.8242	16.9565	16.8753
170	103.9865	20.3321	20.2672
180	132.2781	24.0506	23.8544
190	139.0670	24.8016	24.7305
200	167.3755	27.9641	27.8081

A la vista de los resultados, el efecto de las modificaciones propuestas en esta sección en la Búsqueda Local Rápida no parecen muy significativas sobretodo, si se las compara con el efecto producido por el uso de la propia Búsqueda Local Rápida. Incluso con instancias de menor tamaño la gestión de memoria de las propias variables introducidas hace que el tiempo de computación aumente (aunque de forma poco significativa). Sin embargo, a medida que aumenta el tamaño, el ahorro en el tiempo de computación se va haciendo mayor (tanto de forma relativa como absoluta), especialmente en el primer tipo de instancias, llegando hasta el 7% aproximadamente. Es previsible que para problemas mayores el ahorro aumente aún más.

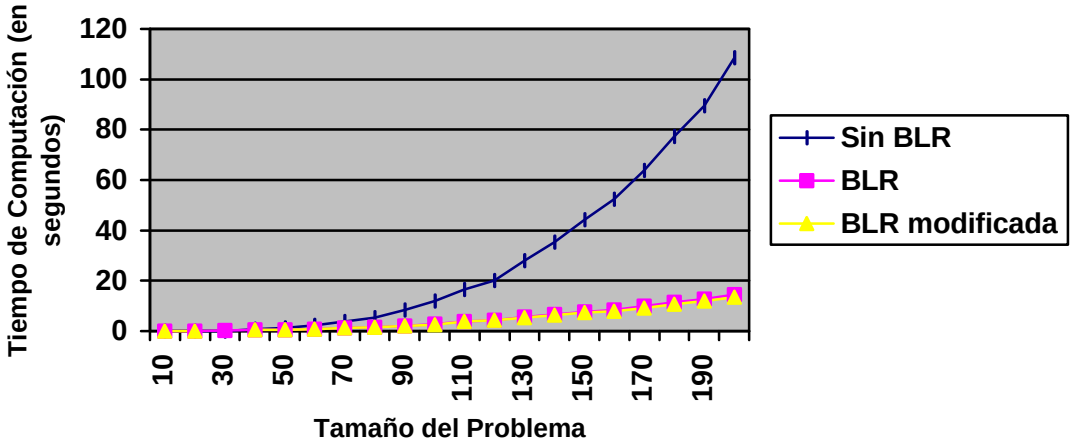

Figura 10.- Evolución de los tiempos medios de computación según el tamaño (instancias tipo I): BLR = Búsqueda Local Rápida.

10 REFLEXIONES FINALES

En este trabajo se ha pretendido analizar algunos aspectos relacionados con el uso de la Búsqueda Local en problemas de rutas. El primero de ellos es la discusión sobre el tipo de vecindario que se construye: es claro que tanto en tiempo de computación como en calidad de los resultados es mejor, en general, considerar intercambios de elementos entre pares de rutas que considerar la solución como una única cadena compacta. En este sentido las mejores soluciones las aporta los CROSS intercambios de Taillard, aunque para problemas de gran tamaño, sería conveniente limitar el tamaño de las cadenas a recolocar.

En segundo lugar también se ha puesto de manifiesto que puede ser conveniente, para problemas de gran tamaño, no tomar la mejor solución vecina, sino la primera

que mejore, puesto que se ahorra casi la mitad del tiempo de computación y la calidad de la solución es parecida.

En tercer lugar es totalmente recomendable el uso de la Búsqueda Local Rápida, o posibles variantes como la analizada en la sección 9, ya que se consiguen reducciones de tiempo espectaculares sin afectar a la solución final. Por otra parte, permite abordar problemas de mayor tamaño que de otra forma no hubiera sido posible, al menos eso se desprende de la evolución de los tiempos de computación. Además, entendemos que esta idea podría ser adaptada fácilmente a otros modelos. En definitiva, el uso adecuado de memoria puede incidir en el ahorro de tiempo de computación.

11 REFERENCIAS Y BIBLIOGRAFÍA

BACKER (DE) B., FURNON V., KILBY P., PROSSER P. and SHAW P. (1.997). "Solving Vehicle Routing Problems using Constraint Programming and Metaheuristics". *Journal of Heuristics*. Vol. 1 - 160.

BENTLEY J.L. (1.992): "Fast Algorithms for Geometric Traveling Salesman Problems". *ORSA Journal on Computing*. Vol. 4, pp. 387-411.

BODIN L.D. and GOLDEN,B.L. (1.981): "Classification in Vehicle Routing and Scheduling". *Networks*. Vol.11, nº 2, 97-108.

BULLHEIMER B., HARTI R.F. and STRAUSS C. (1.997). "Applying the Ant System for the Vehicle Routing Problem". *2nd Metaheuristics International Conference* (MIC-97), Sophie-Antipolis, France, July 1.997.

CAMPOS,V y MOTA, E. (1.995): "Metaheurísticos para el CVRP". *XXII Congreso Nacional de Estadística e Investigación Operativa*. Sevilla, Noviembre 1.995.

CLARKE,G. and WRIGHT,J.W. (1.964): "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points". *Oper.Res*. Vol. 12, pp 568-581.

DESROCHERS, M., LENSTRA, J. K. SAVELSBERGH, M. W. P. and SOUMIS, F. (1.988): "Vehicle Routing with Time Windows: Optimization and Approximation". In *Vehicle Routing: Methods and Studies*, (Studies in Management Sciences and Systems, vol.16), eds: GOLDEN,B.L. and ASSAD,A.A., Nort-Holland, 65-84.

FEO, T. A. and RESENDE, M. G. C. (1.989): "A Probabilistic heuristic for a computationally difficult Set Covering Problem". *Operations Research Letters.* Vol, 8, pp 67-71.

FEO, T. A. and RESENDE, M. G. C. (1.995): "Greedy Randomized Adaptive Search Procedures". *Journal of Global Optimization*. Vol. 2, pp 1-27.

GENDREAU M., HERTZ A. and LAPORTE G. (1.991): "A Tabu Search Heuristic for Vehicle Routing Problem". Report CRT-777. *Centre de Recherche sur les Transports*. Univ. Montréal.

GENDREAU M., HERTZ A. and LAPORTE G. (1.994): "A Tabu Search Heuristic for Vehicle Routing Problem". *Management Sci.*. 40 (10), pp.1276-1290.

GLOVER, F (1.989): "Tabu Search: Part I." *ORSA Journal on Computing*. Vol. 1, pp. 190-206.

GLOVER,F (1.990). "Tabu Search: Part II." *ORSA Journal on Computing*. Vol. 2, pp. 4-32.

Guided Local Search (GLS) Project:
http://cswww.essex.ac.uk/Research/CSP/gls.html.

HAOUARI M., DEJAX P. et DESROCHERS M. (1.990): "Les Problems de Tournées avec Contraintes des Fenênetres de Temps: L'Etat de l'Art". *Recherche Operationnelle/Operations Research.* Vol. 24, N 3, pp 217-244.

KILBY P., PROSSER P. and SHAW P. (1.997). "Guided Local Search for the Vehicle Routing Problem." *2nd Metaheuristics International Conference* (MIC-97), Sophie-Antipolis, France, July 1.997.

KIRPATRICK S., GELATT C. D. and VECCHI M. P. (1.982): "Optimization by Simulated Annealing". *IBM Research Report* RC 9355.

KIRPATRICK S., GELATT C. D. and VECCHI M. P. (1.983): "Optimization by Simulated Annealing". *Science*, Vol. 220, pp 671-680.

KONTORAVDIS,G. and BARD,J.F. (1.995): "A GRASP for the Vehicle Routing Problem with Time Windows". *ORSA Journal on Computing.* Vol. 7, pp 10-23.

LAGUNA, M. (1.999): "Scatter Search". Aparecerá en *Handbook of Applied Optimization*, P. M. Pardalos and M. G. C. Resende (eds). Oxford Academic Press.

LAGUNA, M., FEO, T. and ELROD, H. (1.994): "A Greedy Randomized Adaptive Search Procedures for the 2-Partition Problem". *Operations Research*. Vol. 42, N 4, pp 677-687.

LAGUNA,M., MARTI,R. and CAMPOS,V. (1.998): "Intensification and Diversification with Elite Tabu Search Solutions for the Linear Ordering Problem". Aceptada para su publicación en *Computers & Ops.Res.*

LAPORTE,G. (1.992): "The Vehicle Routing Problem: An overview of exact and approximate algorithms". *European Journal of Operations Research*. Vol. 59, pp 345-358.

LAPORTE, G. and OSMAN, I. H. (1.995). "Routing Problems: A Bibliography". *Ann. Oper. Res.* Vol. 61, pp 227-262.

LIN, S. (1.965): "Computer Solutions to the Traveling Salesman Problem". *Bell Syst. Tech. Jou.* Vol. 44, pp 2245-2269.

LIN, S. y KERNIGHAN, B. W. (1.973): "An Effective Heuristic Algorithm for the Traveling Salesman Problem". *Operations Research. Vol.* 20, pp 498-516.

Memetic Algorithms' Home Page:
http://densis.fee.unicamp.br/~moscato/memetic_home.html.

MOSCATO, P. (1.989): "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms". *Caltech Concurrent Computation Program, C3P Report* 826.

NURMI,K. (1.991): "Traveling Salesman Problem Tools for Microcomputers". *Computers & Ops.Res.* Vol. 18, N. 8, 741-749.

OR, I. (1.976). “Traveling Salesman Type Combinatorial Problems y their Relations to the Logistics of Blood Banking.” *Ph. Thesis.* Dpt. of Industrial Engineering y Management Sciences, Northwestern Univ.

OSMAN, I. H. (1.993): "Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem". *Annals of Operations Research.* Vol. 41, pp 421-451.

PACHECO J. y DELGADO C. (1.997). "Problemas de Rutas con Ventanas de tiempo y carga y Descarga simultánea: Diseño de Filtros para algoritmos de

intercambio (caso de un sólo vehículo)". *Estudios de Economía Aplicada*, nº 7 , pgs. 79-100.

POTVIN,J.Y. and BENGIO,S. (1.994). "A Genetic Approach to the Vehicle Routing Problem with Time Windows". *Technical Report* CRT-953. Centre de Recherche sur les Transports. Univ. Montréal.

POTVIN J. Y., KERVAHUT T., GARCIA B. L. and ROUSSEAU J. M. (1.993)."A Tabu Search Heuristic for Vehicle Routing Problem with Time Windows*". Report CRT-777. Management Sci.* Vol. 40 (10), pp 1276-1290.

REGO, C. (1.998): "A Subpath Ejection Method for the Vehicle Routing Problem". *Management Science.* Vol. 44, N. 10, pp 1447-1459.

ROCHAT, Y. and TAILLARD, E. D. (1.995). "Probabilistic Diversification and Intensification in Local Search for Vehicle Routing". *Journal of Heuristics*. Vol. 1 (1), pp 147-167.

ROSING, K. E. and REVELLE, C. S. (1.997). "Heuristic Concentration: Two Stage solution Construction". *European Journal of Operational Research.* Vol. 97, pp 75-86.

TAILLARD E., BADEAU P., GENDREU M., GUERTAIN F. and POTVIN J.Y. (1.995). "A new Neighborhood structure for the Vehicle Routing Problem with Time Windows". *Technical Repport CRT-95-66. Centre de Recherche sur les Transports*. Univ. Montréal.

TAILLARD, E., BADEAU,P., GENDREU,M., GUERTAIN,F. and POTVIN,J.Y. (1.997). "A Tabu Search heuristic for the Vehicle Routing Problem with Time Windows". *Transportation Science.* Vol. 31, pp 170-186.

THANGIAH S. R., OSMAN I. H., and SUN T. (1.994). "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.

THANGIAH S. R., VINAYAGAMOORTY R., and SUN T. (1.993). "Vehicle Routing Problem with Time Deadlines using Genetic and Local Algorithms". In *5th International Conference on Genetic Algorithms*, 1.993.

VONDOURIS, C. and TSANG, E., (1.999): "Guided Local Search for the Traveling Salesman Problem". *European Journal of Operations Research*. Vol. 113, pp 469-499.

Apéndice 1: Algoritmo para obtener una solución inicial

Se trata de un sencillo algoritmo constructivo que en cada paso inserta un elemento en la solución actual. Sea N = {2, 3, ..., *n*} el conjunto de puntos de visita (carga y descarga). Se define en cada paso:

S = Conjunto de puntos sin insertar en la solución actual;

R_j = Ruta *j-ésima*, j = 2, 3,..., n; (en principio tantas rutas como puntos);

$d(j, i)$ = Incremento de distancia que resulta de insertar *i* en la ruta *j* (la mejor posición); si no existe posición factible donde colocar *j* entonces $d(j,i) = \infty$.

A_i = $d(j^*, i) - d(j(i), i)$

donde: $d(j(i), i)$ = $\min \{d(j, i) / j = 2,...,n\}$

$d(j^*, i)$ = $\min \{d(j, i) / j = 2,...,n; j \neq j(i)\}$

Algoritmo Inicial

Inicialmente hacer S = N; R_j *= 1 - 1 (solamente el origen), j = 2,...,n.*
Repetir
Determinar $A_{i^*} = min \{A_i / i \in S\}$
Insertar i en* $R_{j(i)}$
Ejecutar Busqueda_Local_Or(3,$R_{j(i)}$)*;*
Hacer S = S - {i}*
hasta S = Æ.