QÜESTIIÓ, vol. 26, 3, p. 537-559, 2002

PROBLEMA DE CONTRATACIÓN DE CARRETILLEROS PARA UN ALMACÉN DE PRODUCTOS MANUFACTURADOS

C. R. DELGADO
S. CASADO
J. F. ALEGRE
Universidad de Burgos*

En este trabajo se analiza un problema planteado recientemente a sus autores por una empresa fabricante de componentes de automóviles. Dicha empresa almacena sus productos manufacturados hasta que los clientes (compradores) pasan a recogerlos. Los clientes solicitan sus productos con una frecuencia conocida. Se trata de determinar en función de dichas frecuencias, en qué fechas y a que horas o slots, han de pasar los clientes a recoger sus pedidos. Fijado el horizonte temporal objeto de estudio, el objetivo para la empresa es minimizar en ese conjunto de días, el número de carretilleros necesarios para cargar los pedidos en los camiones de los clientes. El número de carretilleros diario viene determinado por el slot más ocupado. En este problema se han de tomar decisiones a dos niveles: elaboración del calendario de entrega para cada uno de los pedidos, y asignación diaria de pedidos a slots. No se ha encontrado en la literatura ningún trabajo que estudie o pueda ajustarse a este problema. Se diseñan y analizan 3 sencillos Metaheurísticos: uno sigue un proceso de Búsqueda Tabú *básico, otro es un procedimiento de* Búsqueda en Entorno Variable *y el tercero es un* Algoritmo Evolutivo*. Se realizan pruebas con instancias ficticias y por último se resuelven las instancias reales planteadas a los autores de este trabajo.*

Labor requeriments for vehicle loading problem in a store of manufactured products

Palabras clave: Logística, Multiprocessor Scheduling Problem (MSP), Búsqueda Local, Búsqueda Tabú, Búsqueda en Entorno Variable, Algoritmos Evolutivos

Clasificación AMS (MSC 2000): 90B06

* Facultad C. Económicas y Empresariales. Pza. Infanta Elena s/n. 09001 Burgos.

– Recibido en octubre de 2001.
– Aceptado en junio de 2002.

1. INTRODUCCIÓN

Se estudia el caso de una empresa fabricante de componentes de automóviles que almacena sus productos manufacturados a la espera de que los clientes, o compradores, vayan a recogerlos. Los clientes acuden a recoger determinadas cantidades de productos $q(i)$ con una frecuencia conocida. La empresa contrata diariamente los carretilleros necesarios para cargar los productos en los camiones de los clientes, teniendo en cuenta la necesidad del slot (hora) más ocupado.

Se trata de asignar a cada pedido *i* un conjunto de fechas o *calendario* C_r^i −de entre los que tiene disponibles T_i− y en cada día del calendario un slot u hora de entrega; el objetivo es minimizar el número de carretilleros contratados en un determinado horizonte temporal.

Si un pedido tiene una frecuencia de *1 vez cada 4 días* los posibles conjuntos de fechas de recogida de ese pedido, o calendarios, son los siguientes:

$$\Gamma_i = \big\{\{1,5,9,13,17,\ldots\},\quad \{2,6,10,14,18,\ldots\},\quad \{3,7,11,15,19,\ldots\},\quad \{4,8,12,16,20,\ldots\}\big\}$$

(con algunas modificaciones por los días festivos). El horizonte temporal para la planificación es habitualmente de un mes, pero en algunos casos podría ser mayor (2 o 3 meses).

Calendars	Days	1	2	3	4	5	6	7	8	9	10	11	12	13	....
1	{1, 5, 9, 13, .}														....
2	{2, 6, 10, 14, .}														....
3	{3, 7, 11, 15, .}														....
4	{4, 8, 12, 16, .}														....

Figura 1. Posibles calendarios para un producto con una frecuencia de 1 pedido cada 4 días.

El problema implica tomar decisiones a 2 niveles: determinar las fechas para preparar los pedidos, y determinar para cada día las horas de entrega que minimizan el número de carretilleros necesarios. Dicho número, para cada slot, se considera proporcional al número de palets que hay que cargar. Por tanto, sin perdida de generalidad, se puede identificar número de carretilleros necesarios en cada slot con número total de pallets asignados a ese slot. Los contratos de carretilleros son diarios; el número de carretilleros que se contrata diariamente viene determinado por el slot más ocupado. Por tanto, la asignación diaria de slots a pedidos se ajusta al conocido *Multiprocessor Scheduling Problem (MSP)* o al *k-Partition problem.*

Se trata de un problema NP-hard (su problema de decisión asociado es NP-completo). Piénsese que su segundo nivel, el MSP, es un problema NP-completo, como se puede ver por ejemplo en Garey y Johnson (1979).

El MSP es un problema muy estudiado en la literatura y muy estrechamente relacionado con el *Bin Packing*. Algoritmos constructivos para este problema son los conocidos LPT (*largest processing time first*) de Graham (1969) y MULTIFIT de Coffman *et al.* (1978). En este trabajo se utilizará el LPT. Básicamente consiste en ordenar los pedidos de manera decreciente respecto al número de palets que los constituyen, para posteriormente iniciar un proceso de inserción en el slot menos ocupado. Métodos de mejora son los conocidos intercambios de Finn and Horowitz (1979), o el de Langston (1982). Más recientemente en Franca *et al.* (1994) se propone un eficaz heurístico en 3 fases que combina un método constructivo, e intercambios de tipo 0-1 («trasferencia») y 1-1, («intercambio»). Por otra parte en Hübscher and Glover (1994) y Thesen (1998) se proponen estrategias basadas en Búsqueda Tabú. Algoritmos exactos se pueden encontrar en los trabajos de Sahni (1976), Blazewicz (1987), Dell'Amico and Martello (1990) o Harche F. and Seshadri S. (1995). Así mismo en Babel *et al.* (1998) se dan herramientas para su resolución tanto exacta como aproximada. Un reciente «survey» sobre este problema se puede encontrar en Fujita y Yamashita (2000). La dificultad del MSP, en general, aumenta cuando los pedidos son de diferente tamaño, como en el modelo que tratamos.

Una posible formulación del problema que nos ocupa en este trabajo es la siguiente:

$$min \sum_{d=1}^{ndias} \left\{ \max_{j=1\ldots nslots} \sum_{i=1}^{np} q(i) X_{idj} \right\}$$

sujeto a:

$$\sum_{r=1}^{T_i} Y_{ir} = 1 \qquad i = 1,\ldots,np$$

$$\sum_{j=1}^{nslots_i} X_{idj} = Y_{ir} \qquad d \in C_r^i;\ r = 1,\ldots,T_i;\ i = 1,\ldots,np$$

donde:

- *ndias*: número de días;
- *n_slots*: número de slots;
- *np*: número de pedidos;
- $q(i)$: palets del cliente i;
- $\Gamma_i = \left\{C_1^i, C_2^i, \ldots, C_{T_i}^i\right\}$: conjunto de calendarios del pedido i;

– C_r^i: calendario r del pedido i;
– T_i: número de calendarios del pedido i;

con las siguientes variables de decisión:

– Y_{ir}: variable binaria igual a 1 si el pedido i tiene asignado el calendario r y 0 en caso contrario;
– X_{idj}: variable binaria igual a 1 si el pedido i se ha asignado el día d al slot j y 0 en caso contrario.

En principio se trata de un problema particular planteado por una empresa concreta. Sin embargo, entendemos que es un interesante problema, y que problemas muy similares se plantean todos los días a los responsables de diferentes empresas logísticas. No hemos encontrado en la literatura referencias de trabajos que aborden este problema.

Nuestro objetivo será diseñar una estrategia sencilla y adecuada, que dé buenos resultados con datos reales. Así, para la resolución del problema se utilizan tres algoritmos metaheurísticos: uno sigue un procedimiento de *Búsqueda Tabú Básico*, el segundo sigue una estrategia de *Búsqueda en Entorno Variable* (VNS) y el tercero es un *Algoritmo Evolutivo*. Compararemos los resultados obtenidos con cada estrategia, para obtener conclusiones acerca de la eficiencia de cada uno de los tres algoritmos.

El trabajo se estructura como sigue: en el siguiente apartado se propone una forma de representación de soluciones y una definición de movimientos simples entre soluciones; en el tercero se muestra una estrategia basada en un procedimiento muy básico de Búsqueda Tabú, en el cuarto el diseño de un algoritmo basado en Búsqueda en Entorno Variable y en el quinto se describe una estrategia basada en un Algoritmo Evolutivo; en el apartado sexto se muestran los resultados de experimentos computacionales realizados con una serie de instancias ficticias (adaptadas de instancias de la literatura para otros modelos) y otra serie de instancias con datos reales. Finalmente, el último apartado se dedica a las conclusiones.

2. REPRESENTACIÓN DE LAS SOLUCIONES Y DEFINICIÓN DE MOVIMIENTOS

2.1. Representación de soluciones

Las soluciones se representan en dos niveles: en el primero se consideran los calendarios que se asignan a cada pedido, y en el segundo la asignación diaria de pedidos a slots. La asignación de calendarios se representa mediante un vector o *np-upla S*, en el que se indica que calendario corresponde a cada pedido. Por ejemplo, si al pedido p_i le

corresponde el Calendario $C^i_{j_i}$, el correspondiente elemento en el vector es j_i. Por tanto la *np-upla* será de la forma siguiente:

$$S = \left(j_1, j_2, j_3, \ldots, j_{np}\right) / \quad \forall i \in \{1,2,\ldots,np\}, \quad Y_{ij_i} = 1$$

p_1 p_2 p_3 $\ldots$ p_{np} Corresponde al pedido

A la componente *i*-ésima de la solución *S*, la designaremos $S(i)$. Así en *S*, $S(i) = j_i$.

Con **R** se representa, en el conjunto de días, la distribución de pedidos por slots:

$$\mathbf{R} = \{R(d)/d = 1,\ldots,ndias\},$$

siendo $R(d)$ un vector, desde 1 hasta *np*, que indica el slot donde está cada pedido.

Slots	Pedidos $(q(i))$
1	1 -11-5
2	22
3	13-6
4	14-16

Figura 2. Distribución de pedidos para un día «d».

Supongamos un día con cuatro slots u horas de carga: de 9 a 10 (1), de 10 a 11 (2), de 11 a 12 (3) y de 12 a 13 (4); supongamos la distribución de pedidos $R(d)$, propuesta en la figura 2, para un día cualquiera, donde para simplificar la ilustración, se indica de cada pedido *i* directamente la cantidad demandada, es decir $q(i)$. Para cada slot, la suma de palets de cada pedido, indica el número de carretilleros necesarios o nivel de ocupación. El número de carretilleros a contratar cada día viene determinado por el slot más ocupado. En el ejemplo propuesto viene determinado por el slot 4, con dos pedidos de 30 palets en total (es decir 30 carretilleros). El número total de carretilleros a contratar en el conjunto de días se denota como $f(\mathbf{R})$.

2.2. Definición de vecindarios o entornos

Dada una *np-upla S* definimos su vecindario N(S) de la siguiente forma:

$$S' \in N(S) \;\Leftrightarrow\; \exists i' \in \{1,2,\ldots,np\}/S'(i') \neq S(i') \quad \text{y} \quad \forall i \neq i' \quad S'(i) = S(i)$$

es decir, soluciones generadas cuando un pedido (y sólo uno) cambia su *calendario*.

2.3. Movimiento vecinal

El movimiento a una solución vecina puede provocar una disminución parcial Δ, y un aumento parcial Δ' en la función objetivo. La disminución se produce cuando al cambiar de calendario a p_i, este se retira, en alguno de los días, del slot con mayor ocupación; en este caso, debemos calcular y actualizar, en los días que esto suceda, el valor del slot más ocupado. La inserción del pedido p_i en los nuevos días de entrega, se realiza en el slot con menor ocupación, de forma que la función objetivo aumente lo menos posible. Dada una solución $S = \left(j_1, j_2, j_3, \ldots, j_i, \ldots, j_{np}\right)$, para cada pedido $i \in \{1,2,\ldots,np\}$, se denota g_{ij}, a la disminución del valor de la función objetivo, producida al cambiar a p_i, el calendario C_{j_i} por el C_j. Los valores de g_{ij} se calculan como la diferencia $g_{ij} = \Delta' - \Delta$.

Dada la solución (*S*, ***R***), se define:

$$g_{i*j*} = \text{mín}\left\{g_{ij} / i \in \{1,2,\ldots,np\}, \quad j \in \{1,2,\ldots,T_i\}, \quad j \neq j_i\right\};$$

de forma que $i*$ y $j*$ determinan el movimiento a la mejor solución vecina de *S*.

Una vez realizado el movimiento se mejoran los días modificados utilizando un procedimiento de Búsqueda Local. Se proponen dos movimientos para dicho procedimiento −ver figura 3−, usados frecuentemente en múltiples trabajos para problemas combinatorios como por ejemplo, Barnes y Laguna (1993), y para el MSP en Hubscher y Glover (1994), y Thesen (1998):

- Movimiento 0-1: Cambio de un pedido, a un slot diferente al suyo.
- Movimiento 1-1: Intercambio de pedidos, pertenecientes a distintos slots.

Los movimientos 0-1 son fácilmente programables a partir de los movimientos 1-1. Para ello es necesario considerar en cada slot un pedido «dummy» de valor 0.

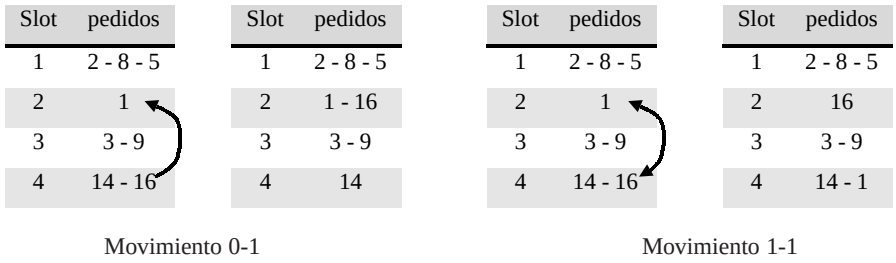

Figura 3. Movimientos vecinales para el MSP.

Para realizar los movimientos Hubscher y Glover (1994) consideran intercambios entre el slot más ocupado y los que tienen una ocupación inferior a la ocupación media; Thesen (1998) considera intercambios entre el slot más ocupado y el menos ocupado. En nuestro caso, ya que tanto el número de pedidos como el de slots no es muy alto, consideramos intercambios entre el slot más ocupado y el resto.

Sea el pedido i perteneciente al slot j, con una ocupación de P_j palets, y el pedido k perteneciente al slot l con una ocupación de P_l palets, la bondad del intercambio entre los pedidos i y k viene dada por:

$$max\{P_j, P_l\} - max\{P_j - q(i) + q(k), P_l - q(k) + q(i)\};$$

nótese que este criterio de bondad es similar a los usados en los trabajos anteriormente señalados. En cada paso se ejecuta el mejor movimiento.

Para acelerar la ejecución de este procedimiento de Búsqueda Local se adapta la estrategia de Búsqueda Local Rápida propuesta por Bentley (1992). Esta estrategia consiste básicamente en lo siguiente: se divide el vecindario, o conjunto de movimientos posibles en subvecindarios; en nuestro caso cada subvecindario, o subconjunto de movimientos, viene definido por un par de slots, es decir, está compuesto por todos los intercambios entre pedidos de ese par de slots. A cada subvecindario se le asocia una variable binaria que indica si está activo o no, de forma que en cada iteración sólo se exploran los subvecindarios activos. Inicialmente todos los subvecindarios están activos. Si tras una exploración se comprueba que en un subvecindario, ningún intercambio de pedidos entre los dos slots que lo componen produce mejora, dicho subvecindario pasa a ser inactivo. Por otro lado, tras la ejecución de un movimiento, todos los subvecindarios alterados, es decir, subvecindarios donde alguno de los slots que los definen se hayan modificado, pasan a ser activos siendo necesario explorarlos de nuevo.

Se han realizado pruebas con problemas generados aleatoriamente cuyo número de pedidos (*np*) asciende a 3.000 y con un número de slots que varía desde 20 hasta 200. Los resultados obtenidos muestran que la Búsqueda Local Rápida reduce el tiempo de computación medio en el conjunto de pruebas en un 12-13 % aproximadamente sin que varíe la solución obtenida. Dicha reducción es mayor cuanto mayor es el número de slots considerado.

2.4. Propuesta de un procedimiento de Búsqueda Local

Teniendo en cuenta lo expuesto hasta ahora, se describe un sencillo algoritmo de búsqueda local para el problema que se analiza en este trabajo:

Procedimiento Búsqueda_Local_Log(var S, **R**)

Repetir
Determinar $g_{i*j*} = min\{g_{ij}/i \in \{1,2,\ldots,np\}, \quad j \in \{1,2,\ldots,T_i\},\ j \neq j_i\}$;
Si $g_{i*j*} < 0$ *entonces:*
– *Hacer* $S(i*) = j*(oj_{i*} = j*)$
– *Modificar y mejorar los elementos de* ***R*** *según se ha explicado en 2.3*
Hasta $g_{i*j*} \geq 0$

3. DISEÑO DE UN ALGORITMO DE BÚSQUEDA TABÚ

En general, en los problemas de optimización combinatoria, la utilización de procedimientos de Búsqueda Local suele llevar a óptimos de mala calidad, debido a la dependencia que existe de la solución inicial. En el problema que estamos tratando se ha observado también esta situación.

El procedimiento de Búsqueda Local puede ser modificado de forma muy sencilla para dar lugar a un procedimiento basado en *Búsqueda Tabú* (muy básico) que puede ayudarnos a escapar de estos pobres óptimos locales, permitiéndonos alcanzar mejores soluciones. La *Búsqueda Tabú* es una estrategia dada a conocer en los trabajos de Glover (1989) y (1990), que está teniendo grandes éxitos y mucha aceptación en los últimos años. Según su creador, es un procedimiento que «*explora el espacio de soluciones más allá del óptimo local*» (Glover y Laguna (1993)). Una vez que se llega a un óptimo local se permiten movimientos *hacia arriba* o que empeoran la solución, para salir de dicho óptimo. Simultáneamente, los últimos movimientos realizados se califican como *tabús* durante las siguientes iteraciones; de esta forma se evita volver a soluciones anteriores y que el algoritmo cicle. Recientes y amplios tutoriales sobre Búsqueda Tabú que incluyen todo tipo de aplicaciones pueden encontrarse en Glover y Laguna (1997) y (2001).

En nuestro caso se define:

Matriz_tabú $(i,\ j)$ = número de la iteración en la que el pedido p_i dejó de tener asignado el calendario C_j^i.

Se denota por (S, **R**) a la solución actual y por (S*, **R***) a la mejor solución encontrada; el valor de la función objetivo para dichas soluciones será, respectivamente, f y $f*$; el algoritmo de Búsqueda Tabú se puede escribir de la forma siguiente:

Algoritmo Búsqueda_Tabú_Log(var S, **R**; S*, **R***)

Hacer Matriz_tabú $(i,j) = -Tabu\_Tenure, i \in \{1,2,\ldots,np\}, \quad j \in \{1,2,\ldots,T_i\}$;
Hacer niter = 0, iter_mejor = 0, S = S, y* $\boldsymbol{R^*} = \boldsymbol{R}$
Repetir
  niter = niter + 1
  Determinar
    $g_{i*j*} = min\{g_{ij}/i \in \{1,2,\ldots,np\}, j \in \{1,2,\ldots,T_i\}; j \neq j_i$ *verificando que*
      [niter >
    Matriz_tabú $(i,j) + Tabu\_Tenure$ *o* $\quad f + g_{ij} < f*$ *(«criterio de aspiración»)]}*
  Hacer Matriz_tabú $(i*, j_{i*}) = niter$ *y* $\quad S(i*) = j*(oj_{i*} = j*)$
  Modificar y Mejorar los elementos de **R** *según se explicó en 2.3*
  Actualizar f
  Si $f < f*$ *entonces hacer:* $S* = S$, $\boldsymbol{R^*} = \boldsymbol{R}$, $f* = f$ *e iter_mejor = niter*
Hasta (niter $-$ *iter_mejor* $>$ *max_iter) u otro criterio de parada*

El parámetro *Tabu_Tenure* indica el número de iteraciones en las que no se permitirá a un pedido volver a tener asignado un calendario que acaba de dejar. Esta restricción o *condición tabú* puede ser superada por el *criterio de aspiración*: una solución o movimiento tabú puede ser aceptado, si da lugar a una solución mejor que la mejor encontrada hasta el momento. El parámetro *max_iter* indica el máximo número de iteraciones sin mejoras.

4. DISEÑO DE UN ALGORITMO BASADO EN BÚSQUEDA EN ENTORNO VARIABLE

Búsqueda en Entorno Variable (*Variable Neighborhood Search* o VNS) es un reciente metaheurístico diseñado para resolver problemas combinatorios, propuesto y descrito 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 sistemático cambio del entorno de búsqueda. El algoritmo aplica la búsqueda local en alguna solución del entorno 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. Recientes tutoriales con ejemplos se encuentran en Hansen y Mladenovic (2000) y (2001).

En nuestro caso para obtener una solución en un entorno de la solución actual se perturba esta de la siguiente manera. Sea (S, **R**) una determinada solución, y $k \in$ IN se define:

Procedimiento Perturbar (k,S)

Seleccionar k pedidos aleatoriamente, $p_{i1}, p_{i2}, \ldots, p_{ik}$, *con* $T_{il} \geq 2, l = 1 \ldots k$,
Para $l = 1, \ldots, k$:
Reasignar a p_{il} *un calendario* $C_{l'}^{il} \in \Gamma_i / l' \neq S(il)$ *escogido aleatoriamente*

Sea (S, $\boldsymbol{R}$) una solución con valor f, el algoritmo VNS que se propone en este trabajo queda de la forma siguiente:

Algoritmo VNS (*kmax, S,* $\boldsymbol{R}$)

Hacer niter = 0
Repetir
Hacer: niter = niter + 1 y k = 0
Repetir
Hacer k = k+1;
$S' = S$
Ejecutar Perturbar (k,S')
Construir R' : $\forall d = 1, \ldots,$ *ndias formar* $\boldsymbol{R}(d)$ *con un algoritmo constructivo para el MSP (LTP de Graham* et al. *(1969)) y mejorar con intercambios 0-1,1-1*
Determinar f' *el valor de* $(S', \boldsymbol{R}')$
Ejecutar Búsqueda_Local_Log $(S', \boldsymbol{R}')$
Hasta $(f' < f)$ *o (k = kmax)*
Si $f' < f$ *hacer:* $S = S', R = R'$ *y* $f = f'$*e iter_mejor = niter*
Hasta (niter-iter_mejor = maxiter) u otro criterio de parada

El parámetro *kmax* indica el máximo número de puntos que se van a cambiar y *max_iter* el máximo número de iteraciones sin mejora. El uso de un procedimiento de perturbación aleatorio tiene como objeto evitar ciclos.

5. PROPUESTA DE UN ALGORITMO EVOLUTIVO

Los Algoritmos Evolutivos se inspiran en los principios básicos de la evolución de los seres vivos y los modifican 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 *poblaci ón*. 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 bondad (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. El algoritmo evolutivo diseñado, emplea heurísticos para mejorar las soluciones antes de recombinarlas, introduciendo así, un mayor conocimiento del problema.

En nuestro caso, el Algoritmo Evolutivo sigue el siguiente esquema básico:

Algoritmo Evolutivo_Log

Generar una población inicial de soluciones
Mejorarlas mediante un método preestablecido
Repetir
- *Seleccionar aleatoriamente un subconjunto de elementos (par) de la población con una probabilidad proporcional a su bondad*
- *Cruce Reproducción: Emparejar o cruzar estas soluciones (Padres) para dar lugar a nuevas soluciones (Hijos). De cada pareja de padres se ha de generar una nueva pareja de hijos*
- *Mutación: Las soluciones hijas pueden cambiar, con una pequeña probabilidad, alguno de sus elementos (genes)*
- *Aplicar el procedimiento de mejora a los hijos*
- *Sustituir las peores soluciones de la población por las nuevas soluciones hijas*

Hasta conseguir algún criterio de parada

A continuación se hacen las siguientes consideraciones. Las operaciones de cruce y mutación se realizan sobre los vectores S de cada solución. El operador de cruce sigue el esquema «one-point crossing», es decir, sean S y S' dos vectores «padres» seleccionados:

$$S = (j_1, j_2, j_3, \ldots, j_{np}) \quad \text{y} \quad S' = (j'_1, j'_2, j'_3, \ldots, j'_{np})$$

se genera aleatoriamente un «punto de cruce» (pto_cruce) entre 1 y np, de forma que los nuevos vectores hijos se definen como:

$$\begin{aligned} S'' &= (j_1, j_2, \ldots, j_{pto\_cruce}, j'_{pto\_cruce+1}, \ldots, j'_{np}) \\ S''' &= (j'_1, j'_2, \ldots, j'_{pto\_cruce}, j_{pto\_cruce+1}, \ldots, j_{np}). \end{aligned}$$

Los componentes (calendarios) de cada nuevo vector hijo pueden cambiar o *mutar* con una pequeña probabilidad, *p_mut*. Para decidir si un determinado componente cambia se genera un valor aleatorio uniformemente en (0, 1); si este valor es menor que *p_mut* se realiza el cambio. Este consiste en elegir aleatoriamente un calendario diferente al actual y asignárselo (*mutación*). Este proceso de mutación tiene por objeto dar diversidad al proceso y evitar que este se encajone en una región entorno a un mínimo local.

Una vez que se genera un nuevo vector *S* y pasa por el proceso de mutación, la segunda componente de la solución, $\boldsymbol{R}$, se define de la forma siguiente: $\forall d = 1, \ldots,$ *ndias* se consideran todos los pedidos que se han de entregar el día *d*, según lo establecido por *S*; la composición de los slots de $\boldsymbol{R}(d)$ se genera de acuerdo al siguiente esquema (al igual que en el apartado anterior con el algoritmo VNS):

- Diseño de $\boldsymbol{R}(d)$ mediante el algoritmo *LTP de Graham* et al. *(1969) para el MSP.*
- Mejora con intercambios 0-1 y 1-1, según se expuso en el subapartado 2.3.

Una vez generada $\boldsymbol{R}$, la solución hija (*S*, $\boldsymbol{R}$), es mejorada aplicando el procedimiento *Búsqueda_Local_Log*(S, $\boldsymbol{R}$).

Se definen los siguientes parámetros: *n_pob* = número de elementos de la población, *n_sel* = número de padres seleccionados, *p_mut* = probabilidad de mutación, y *max_gen* = número de «generaciones» (iteraciones) sin mejora como criterio de parada.

6. EXPERIENCIAS COMPUTACIONALES

Para contrastar y comparar la eficacia de los 3 procedimientos propuestos en este trabajo se han realizado una serie de pruebas, tanto con instancias ficticias (adaptadas de instancias de la literatura para otros problemas), como con instancias tomadas de los datos reales facilitados.

Se han tomado de adaptaciones de las instancias número 1, 2, 5 y 10 Christofides y Beasley, (1984) (instancias para el PVRP, *Periodic Vehicle Routing Problem*, correspondientes a *np* = 51, 50, 75 y 100), para 30, 60 y 90 días, (pues son tiempos habituales para los que se realizan las planificaciones de producción) con *n_slots* = 4 −media jornada− y 8 −jornada completa−. (En el apéndice se describe la forma en que se han adaptado estas instancias a nuestro problema).

Para cada caso se ha generado una población de 100 soluciones aleatorias [1]. Esta población sirve de punto de partida del algoritmo Evolutivo −Evol−. Por otra parte, la mejor

[1] Asignación de calendarios aleatoria y composición de slots, en cada día, usando LTP y mejorando con intercambios 0-1, 1-1.

de estas soluciones (*SI*) sirve de punto de partida para los otros 2 algoritmos, Búsqueda Tabú −BT− y VNS. En el caso de VNS, dado que la ejecución no es determinista, se presentan los resultados medios de 10 ejecuciones para cada instancia. Los parámetros han utilizado los siguientes valores: *Tabu_Tenure* = $np \cdot ndias$ (Búsqueda Tabú), *pmut* = 0.05, *npob* = 100 y *nsel* = 20 (Algoritmo Evolutivo). Como criterios de parada se han utilizado los siguientes:

- En el caso de la Búsqueda Tabú el criterio de parada es que pasen ($20 \cdot np \cdot n\_dias$) iteraciones sin mejora (*max_iter* = ($20 \cdot np \cdot n\_dias$)).
- En el caso de Búsqueda en el Entorno Variable es que *kmax* (número máximo de pedidos que se seleccionan para perturbar) = 20, y *max_iter* (número de iteraciones sin mejora) = *np*.
- Para el Algoritmo Evolutivo *max_gen* = *np* (generaciones sin mejorar).

Los resultados obtenidos para las instancias ficticias cuando *n_slots* = 4 son los que se muestran en la tabla 1 (en negrita se resalta la mejor solución).

Tabla 1. Resultados de las instancias ficticias (n_slots=4).

Instancia	n_dias	SOL. INICIAL	BT	VNS	EVOL
			Tiempo de Computación Total/T.C. hasta encontrar la mejor solucion		
1	30	1097	1010 ***14,89/0,00***	1009,7 *47,52/22,00*	**1007** *49,6/35,43*
1	60	1800	1696 *114,85/**56,96***	1686,1 ***107,48**/71,48*	**1676** *137,09/113,64*
1	90	2648	2545 ***75,19/0,05***	2531,7 *126,95/79,74*	**2511** *161,1/125,29*
2	30	953	**945** *28,06/13,45*	946,3 *36,42/**12,16***	946 ***27,13**/13,73*
2	60	1636	1619 *106,5/49,98*	**1617,8** *77,82/35,81*	1620 ***32,52/9,29***
2	90	2451	**2420** *69,53/**0,16***	2421,9 *107,63/51,23*	2425 ***39,93**/6,15*
5	30	1747	**1742** *40,86/6,81*	1742,6 *93,12/29,28*	1743 ***38,5/9,78***
5	60	2958	**2951** *339,66/211,74*	2952,7 *157,54/57,36*	2954 ***58,94/9,29***
5	90	4501	4491 *239,36/77,77*	**4488,2** *218,42/83,80*	4489 ***107,6/34,05***
10	30	1501	**1500** *88,59/23,95*	1500,05 *107,76/17,98*	1501 ***38,89/0,00***
10	60	2475	**2462** *285,07/36,69*	2463,4 *282,82/125,80*	2467 ***90,9/33,06***
10	90	3755	3738 *426,99/**100,89***	**3734,1** *514,51/305,75*	3744 ***171,48**/ 104,41*
TOTALES		27522	27119 *1829,55/578,45*	27094,55 *1878,02/892,43*	**27083** ***953,68/494,12***

Los resultados obtenidos para las instancias ficticias cuando *n_slots* = 8 son los que se muestran en la tabla 2.

Tabla 2. Resultados de las instancias ficticias (n_slots = 8).

Instancia	n_dias	SOL. INICIAL	BT	VNS	EVOL
			Tiempo de Computación Total/T.C. hasta encontrar la mejor solucion		
1	30	763	616 ***32,52/16,20***	622,4 *53,47/30,18*	**610** *64,48/48,44*
1	60	1215	**981** ***60,75/0,77***	998,3 *69,68/35,05*	992 *70,8/40,65*
1	90	1705	1450 *83,7/**6,15***	1480,6 ***81,49**/33,95*	**1444** *128,91/85,62*
2	30	613	**515** ***31,03/15,1***	520 *58,63/26,73*	518 *48,11/32,19*
2	60	1037	888 ***63,83/0,61***	892,8 *122,89/69,85*	**866** *119,46/94,8*
2	90	1531	**1329** ***71,73/0,99***	1342,1 *126,01/70,14*	1358 *71,96/37,74*
5	30	912	**887** ***51,85/0,05***	889,6 *110,47/23,78*	893 *58,22/19,33*
5	60	1612	1516 *178,95/**0,88***	1522,1 *193,71/68,12*	**1515** ***153,25**/91,35*
5	90	2393	2299 *238,7/**0,49***	2308,8 ***220,03**/69,78*	**2295** *391,07/300,33*
10	30	818	**760** *86,73/**1,48***	761,4 *186,5/64,82*	763 ***70,31**/25,21*
10	60	1358	**1257** *323,89/**2,19***	1261,1 *421,97/234,62*	1264 ***205,59**/130,56*
10	90	2019	**1907** *435,51/**9,12***	1911,9 *576,44/54,26*	1933 ***97,16**/26,09*
TOTALES		15976	**14405** *1659,19/**54,03***	14511,1 *221,30/1081,31*	14451 ***1479,32**/932,31*

Búsqueda Tabú aporta los mejores resultados para un mayor número de instancias. En el caso de las instancias de menor número de pedidos (1 y 2), las mejores soluciones las aporta, en general, el Algoritmo Evolutivo, mientras que en las instancias de mayor número de pedidos (5 y 10) las mejores soluciones las aporta Búsqueda Tabú. Con respecto al tiempo de computación, se observa que el Algoritmo Evolutivo es en general, el que menos tiempo de computación total emplea para aportar su mejor solución, aunque en las instancias de menor tamaño y sobre todo para *n_slots* = 8 BT aporta soluciones más rápidamente. VNS es el que peores soluciones aporta, sobre todo en el caso de *n_slots* = 8.

Dada la dificultad para obtener la solución óptima, se han obtenido unas, entendemos que, buenas aproximaciones a dicho óptimo. Para ello se han ejecutado todos los algoritmos para cada instancia, con diferentes soluciones y poblaciones iniciales, y un

tiempo de parada de una hora, y se ha tomado la mejor solución. Los resultados se muestran en la tabla 3.

Tabla 3. Instancias ficticias. Comparación con la mejor solución encontrada.

n_solts	Inst.	np	n_dias	BT	VNS	EVOLU.	Mejor Sol. Conocida	% Desviac.
4	1	51	30	1010	1009,7	**1007**	**1001**	0,60
			60	1696	1686,1	**1676**	**1668**	0,48
			90	2545	2531,7	**2511**	**2495**	0,64
	2	50	30	**945**	946,3	946	**937**	0,85
			60	1619	**1617,8**	1620	**1599**	1,18
			90	**2420**	2421,9	2425	**2396**	1,00
	5	75	30	**1742**	1742,6	1743	**1737**	0,29
			60	**2951**	2952,7	2954	**2940**	0,37
			90	4491	**4488,2**	4489	**4467**	0,47
	10	100	30	**1500**	1500,05	1501	**1494**	0,40
			60	**2462**	2463,4	2467	**2452**	0,41
			90	3738	**3734,1**	3744	**3721**	0,35
8	1	51	30	616	622,4	**610**	**574**	6,27
			60	**981**	998,3	992	**938**	4,58
			90	1450	1480,6	**1444**	**1392**	3,74
	2	50	30	**515**	520	518	**499**	3,21
			60	888	892,8	**866**	**856**	1,17
			90	**1329**	1342,1	1358	**1289**	3,10
	5	75	30	**887**	889,6	893	**880**	0,80
			60	1516	1522,1	**1515**	**1498**	1,13
			90	2299	2308,8	**2295**	**2264**	1,37
	10	100	30	**760**	761,4	763	**755**	0,66
			60	**1257**	1261,1	1264	**1237**	1,62
			90	**1907**	1911,9	1933	**1877**	1,60

Si comparamos la mejor solución aportada por los algoritmos iniciales con la «*Mejor Solución Conocida*» vemos que se consiguen resultados satisfactorios para $n\_solts = 4$ (la desviación es sólo del 0,58 %) y aceptables para $n\_solts = 8$ (la desviación es del 2,43 %).

Finalmente se va a analizar el comportamiento de las estrategias desarrolladas en este trabajo, para la resolución de una serie de instancias generadas a partir de los datos reales facilitados por los responsables de la empresa fabricante de componentes de automóviles. Se tienen 45 pedidos con las frecuencias y cargas que se indican en la tabla 4.

Tabla 4. Datos de los pedidos.

nº ped	*freq*	*q*	*nº ped*	*freq*	*q*	*nº ped*	*freq*	*q*
1	1D	12	**16**	1M	1	**31**	1S	2
2	1M	1	**17**	1D	15	**32**	1S	6
3	1M	2	**18**	1M	2	**33**	1M	4
4	1M	1	**19**	2S	4	**34**	1M	1
5	2S	20	**20**	2S	6	**35**	1D	3
6	1D	3	**21**	2M	3	**36**	1D	8
7	1S	3	**22**	1S	1	**37**	2M	2
8	1M	1	**23**	1S	2	**38**	2S	3
9	1D	6	**24**	2M	1	**39**	1S	2
10	1S	6	**25**	2M	1	**40**	1S	8
11	1M	1	**26**	1M	1	**41**	1S	10
12	1M	3	**27**	2S	6	**42**	1M	1
13	1S	8	**28**	2S	12	**43**	2S	6
14	1S	8	**29**	1S	1	**44**	1S	10
15	1D	3	**30**	1M	1	**45**	1S	2

El significado de las frecuencias es el siguiente: 1D = una recogida cada día; 1S = una recogida a la semana; 2S = una vez cada 2 semanas; en el primer caso sólo hay un calendario posible $\{(1,2,3,\ldots,ndias)\}$; en el segundo el conjunto de calendarios sería de la siguiente forma $\{(1,8,15,22,29,\ldots),\ (2,9,16,23,30,\ldots),\ \ldots\}$; en el tercero el conjunto sería $\{(1,15,29,\ldots),\ (2,16,30,\ldots),\ \ldots\}$.

Para estos datos se consideran horizontes de 30, 60 y 90 días y *n_slots* = 4 (media jornada) y 8 (jornada completa), como en las anteriores instancias. Suponemos que el día 1 es lunes y se van a considerar festivos los domingos (es decir, los días 7, 14, 21, 28, ...). Si al diseñar un calendario algún día resulta ser festivo se cambia por el siguiente día (día 8 en vez de día 7, por ejemplo), sin variar el resto del calendario.

Para la resolución de estos problemas reales, Búsqueda Tabú (BT) y VNS, usan como solución inicial $-SI-$ la aportada por una empresa logística[2], mientras que el Algoritmo Evolutivo $-$Evol$-$ usa una población de soluciones iniciales aleatorias como en las instancias anteriores.

Los resultados obtenidos para la instancia real son los que aparecen en la tabla 5 (en negrita la mejor solución). El tamaño del problema es en este caso similar al de las

[2]Grupo SAAT, (Sociedad Anónima de Aduanas y Transportes), Aduana Interior Villafría, 09192, Burgos, Spain.

instancias 1 y 2 anteriores. Al igual que en dichas instancias el mejor resultado lo aporta el Algoritmo Evolutivo.

Tabla 5. Resultados para las instancias reales.

slots	n_dias	SOLUCIÓN INICIAL	BT	VNS	EVOL
			Tiempo de Computación Total/T.C. hasta encontrar la mejor solucion		
4	30	516	448 ***10,38/2,75***	443,6 *20,68/7,96*	**440** *24,67/17,58*
4	60	999	892 *51,73/19,71*	889,1 *34,17/12,96*	**887** ***22,52/8,51***
4	90	1484	1338 *96,06/25,21*	1336,8 *61,91/30,86*	**1332** ***46,69/26,31***
8	30	419	**400** *8,29/0,22*	**400** *12,93/0,19*	**400** ***5,77/0,00***
8	60	827	**800** *31,31/0,44*	**800** *20,02/0,31*	**800** *10,55/0,00*
8	90	1235	**1200** *68,16/0,71*	**1200** *21,67/0,42*	**1200** *14,94/0,00*

Podemos afirmar teniendo en cuenta estos resultados lo siguiente:

- Los tres algoritmos propuestos consiguen mejorar significativamente las soluciones propuestas por la empresa logística.
- Entre los tres algoritmos no hay ninguna diferencia para $n\_slots = 8$ (jornada completa), sin embargo si que se aprecian algunas diferencias para $n\_slots = 4$ (media jornada).
- En este último caso, el Algoritmo Evolutivo consigue llegar a soluciones mejores que los otros dos. Búsqueda Tabú es el que aporta peores soluciones.

7. CONCLUSIONES

En este trabajo se ha analizado un problema real planteado a los autores de este trabajo por los responsables de una fábrica de componentes de automóviles.

Se trata de un problema a dos niveles: en el primer nivel se trata de asignar un calendario de recogida a cada pedido; en el segundo nivel se trata de asignar, en cada día, una hora de recogida a cada pedido. El objetivo es racionalizar (minimizar) el número de contratos del personal encargado de preparar y cargar los pedidos periódicos que realizan los clientes en los vehículos de dicha empresa (carretilleros).

Del problema conjunto no existen referencias previas anteriores; aunque el segundo nivel esta muy relacionado con el *Multiprocessor Scheduling Problem (MSP)*, problema muy estudiado en la literatura y relacionado con el *Bin Packing*.

Para su resolución se proponen tres sencillos algoritmos Metaheurísticos basados en movimientos vecinales: uno sigue un procedimiento de *Búsqueda Tabú* básico, el segundo sigue una estrategia de *Búsqueda en Entorno Variable* (VNS), y el tercero es un *Algoritmo Evolutivo.*

Búsqueda Tabú consigue para un mayor número de instancias mejores soluciones que los otros dos. Se observa sin embargo, que las soluciones aportadas por el Algoritmo Evolutivo son de mayor calidad que las de Búsqueda Tabú para las instancias de menor tamaño; en las instancias de mayor tamaño ocurre justamente lo contrario.

Finalmente las estrategias analizadas mejoran sustancialmente para los problemas basados en datos reales, las propuestas realizadas por la empresa logística con la que trabaja habitualmente la fábrica de componentes de automóviles.

En cualquier caso, este trabajo supone una primera aproximación, ya que las estrategias analizadas son muy básicas. En futuros trabajos se espera completarlas con otros elementos (memoria a medio y largo plazo, renovación de la población), además de analizar otras estrategias u otros tipos de movimientos vecinales.

8. APÉNDICE: GENERACIÓN DE INSTANCIAS

Como se ha comentado en el apartado 6, se han generado diferentes instancias para este problema tomando los datos de conocidas instancias para el VRP o PVRP. La forma de realizar las adaptaciones es la siguiente:

- Se leen de dichas instancias las unidades o pallets $-q(i)$, $i = 1, \ldots, np-$ demandadas de los pedidos;
- Se ordenan los pedidos según los valores de $q(i)$, de menor a mayor, para agruparlos: en el primer grupo los de menor valor de q, en el segundo los siguientes, ..., y en el último los de mayor valor.
- Se asigna la misma frecuencia y calendarios a todos los pedidos del mismo grupo.
- Se generan 3 instancias por cada instancia original del VRP o PVRP leída, correspondientes a horizontes temporales de 30, 60 y 90 días.
- Para las instancias de 30 días se consideran 6 grupos: al primer grupo se le asigna una frecuencia de entrega diaria, (i.e., 30 entregas), al segundo una entrega cada 3 días (10 entregas), al tercero cada 5 días (6 entregas), al cuarto cada 10 días, al quin-

to cada 15 días y al sexto cada 30 días (una única entrega).

– Para las instancias de 60 días se toman 7 grupos y las siguientes frecuencias de entrega: 1 día (60 entregas), 2 (30), 6, 10, 20, 30 y 60.
– Para las instancias de 90 días se toman 8 grupos y las siguientes frecuencias de entrega: 1 día (90 entregas), 2 (45), 3, 6, 10, 30, 45 y 60.

Una vez que se asignan las frecuencias, los calendarios se generan automáticamente; por ejemplo, en un horizonte de 30 días los pedidos que tengan una entrega cada 3 días (segundo grupo), tendrán el siguiente conjunto de calendarios: $\{1,4,7,10,13,16,\ldots,28\}$, $\{2,5,8,11,14,17,\ldots,29\}$ y $\{3,6,9,12,15,18,\ldots,30\}$.

9. BIBLIOGRAFÍA

Babel, L., Kellerer, H. and Kotov, V. (1998). «The k-partitioning Problem», *Mathematical Methods of Operations Research*, 47 (1), 59-82.

Barnes, J. W. and Laguna, M. (1993). «A Tabu Search Experience in Production Scheduling», *Annals of Operations Research*, 41, 141-156.

Bentley, J. L. (1992). «Fast Algorithms for Geometric Traveling Salesman Problems», *ORSA Journal on Computing*, 4, 387-411.

Blazewicz, J. (1987). «Selected Topics in Scheduling Theory», *Annals Discr. Math.*, 31, 1-60.

Coffman Jr., E. G., Garey, M. R. and Johnson, D. S. (1978). «An Application to the Bin-Packing to Multiprocessor Scheduling», *SIAM J. Comput.*, 7, 1-17.

Christofides, N. and Beasley, J. E. (1984). «The Period Routing Problem», *Networks*, 14, 237-256.

Dell' Amico, M. and Martello, S. (1990). «Optimal Scheduling of Tasks on Identical Parallel Processors», Research Report OR/90/7. *DEIS*, University of Bologna.

Finn, G. and Horowitz, E. (1979). «A Linear Time Approximation Algorithms for Multiprocessor Scheduling», *BIT*, 19, 312-320.

Franca, P., Gendrau, M., Laporte, G. and Muller, F. M. (1994). «A Composite Heuristic for the Identical Parallel Machine Scheduling Problem with Minimun Makespan Objective», *Computers Ops. Res.*, 21, 2, 205-210.

Fujita, S. and Yamashita, M. (2000). «Approximation Algorithms for Multiprocessor Scheduling Problem», *IEICE Trans. Inf. & Syst.*, E83-D, 3, 503-509.

Garey, M. R. y Johnson, D. S. (1979). «Computers and Intractability». Freeman.

Graham, R. L. (1969). «Bounds on Multiprocessing Timing Anomalies», *SIAM J. Appl. Math.*, 17, 416-429.

Glover, F. (1989). «Tabú Search: Part I», *ORSA Journal on Computing*, 1, 190-206.

Glover, F. (1990). «Tabú Search: Part II», *ORSA Journal on Computing*, 2, 4-32.

Glover, F. (1996). «Búsqueda Tabú», en *Optimización Heurística y Redes Neuronales*. Adenso Díaz (coordinador). Paraninfo. Madrid, 105-143.

Glover, F. y Laguna, M. (1993). «Tabu Search», in *Modern Heuristic Techniques for Combinatorial Problems*. C. Reeves, ed., Blackwell Scientific Publishing, 70-141.

Glover, F. y Laguna, M. (1997). «Tabu Search». Kluver Academic Publishers, Boston.

Glover, F. y Laguna, M. (2002). «Tabu Search», *Handbook of Applied Optimization*, P. M. Paradalos and M. G. S. Resende (eds). Oxford University Press, 194-208.

Hansen, P. and Mladenovic, N. (1998). «An Introduction to Variable Neigborhood Search», in S. Voss *et al.* (eds.), *Metaheuristics Advances and Trends in Local Search Paradigms for Optimization*, 433-458, MIC-97, Kluwer, Dordrecht.

Hansen, P. and Mladenovic, N. (2000). «Recherche a Voisinage Variable», *Les Cahiers de GERAD*, G-2000-08, Montreal, Canada.

Hansen, P. and Mladenovic, N. (2001). «An Introduction to Variable Neigborhood Search». To appear in *Handbook of Applied Optimization*, P. M. Pardalos and M. G. C. Resende (eds.), Oxford Academic Press.

Harche, F. and Seshadri, S. (1995). «An LPT-Bound for a Parallel Multiprocessor Scheduling Problem», *Journal of Mathematical Analysis and Applications*, 196 (1), 181-195.

Hubscher, R. and Glover, F. (1994). «Applying Tabu Search with Influential Diversification to Multiprocessor Scheduling», *Computers and Operations Research*, 21, 877-844.

Langston, M. A. (1982). «Improved 0/1 Interchange Scheduling», *BIT*, 22, 282-290.

Mladenovic, N. (1995). «A Variable Neighborhood Algorithm-A New Metaheuristic for Combinatorial Optimization». Abstracts of papers presented at *Optimization Days*, Montreal, 112.

Mladenovic, N. and Hansen, P. (1997). «Variable Neighborhood Search», *Computers and Operations Research*, 24, 1097-1100.

Moscato, P. (1989). «On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms», *Caltech Concurrent Computation Program*, C3P Report 826.

Sahni, S. (1976). «Algorithms for Scheduling Independent Tasks», *J. ACM*, 23, 1, 116-127.

Thesen, A. (1998). «Design and Evaluation of Tabu Search Algorithms for Multiprocessor Scheduling», *Journal of Heursitics*, 4, 141-160.

ENGLISH SUMMARY

LABOR REQUERIMENTS FOR VEHICLE LOADING PROBLEM IN A STORE OF MANUFACTURED PRODUCTS

C. R. DELGADO
S. CASADO
J. F. ALEGRE
Universidad de Burgos*

This paper examines a problem proposed by a local autopart enterprise. This firm stores its outputs until the customers go to collect them. The clients apply for their products with a known frequency. The problem is to determine when the clients have to pick up their orders (the dates and the hours or slots).
With a planning horizon fixed, the aim of the firm is to minimize, for the joint of days, the number of «wheelbarrow mans» that are necessary for loading the orders in the customers' trucks; this number is determined each day for the busiest slot.
In this problem the decisions are made in two levels: doing a calendar of delivery for each order and the daily assignation of slots to the orders. There isn't any work in the literature that studies a problem like this.
In this work are designed and analyzed three simple Metaheuristics Procedures: the first is a basic Tabu Search, the second is an algorithm based on Variable Neighborhood Search and the third is a Evolutionary Algorithm. Computational experiences with fictitious and real instances are made.

Keywords: Logistic, Multiprocessor Scheduling Problem (MSP), Local Search, Tabu Search, Variable Neighborhood Search, Evolutionary Algorithm

AMS Classification (MSC 2000): 90B06

* Facultad C. Económicas y Empresariales. Pza. Infanta Elena s/n. 09001 Burgos.

– Received October 2001.
– Accepted June 2002.

In this paper, we address a logistical problem that a manufacturer of auto parts described to the authors. The manufacturer stores the products in its warehouse until customers pick them up. The customers and the manufacturer agree upon an order pickup frequency. For example, a customer may require a frequency of one pickup every other day, while another customer may require a frequency of one pickup every 4 working days. If the required frequency is of one visit to the manufacturer every 4 working days, then the possible pickup calendars in a 20-day planning horizon are:

$$\{1, 5, 9, 13, \text{ and } 17\}$$
$$\{2, 6, 10, 14, \text{ and } 18\}$$
$$\{3, 7, 11, 15, \text{ and } 19\}$$
$$\{4, 8, 12, 16, \text{ and } 20\}$$

where the numbers indicate the index of the day when the customer visits the manufacturer.

The problem is to find the best pickup schedule, which consists of the days and times during the day that each customer is expected to pick up his/her order. For a given planning horizon, the optimization problem is to minimize the labor requirements to load the vehicles that the customers use to pick up their orders. We approach this situation as a decision problem with two levels:

1. To assign each customer to a set of days in the planning horizon such that the frequency requirements are satisfied. That is, the problem is to assign each customer to one of its feasible pickup calendars.
2. To assign each customer to a time slot within each day such that the labor requirements are minimized.

Clearly, the decisions in the first sub-problem condition the best possible solutions that can be found when solving the second sub-problem. That is, the second sub-problem tries to minimize the labor requirements in each day, but the set of customers assigned to each day constraints the solution space. While, to the best of our knowledge, the entire problem has not been addressed in the literature, the second sub-problem is equivalent to the *multiprocessor scheduling problem* (MSP), or also known as the *k partition problem*, and to the *bin packing problem* that seeks to minimize the heaviest bin.

The objective function minimizes the sum of the daily labor requirements in the planning horizon. The maximum number of workers needed in any slot gives the labor requirements for the day. In the environment that we studied, workers cannot be hired on an hourly basis. That is, a full day is the minimum amount of time that a worker can be hired to load auto parts onto customer vehicles.

In this work are designed and analyzed three simple Metaheuristics Procedures: the first is a basic *Tabu Search*, the second is an algorithm based on *Variable Neighborhood Search* and the third is an *Evolutionary Algorithm*. Computational experiences with fictitious and real instances are performed.

Tabu Search arises the best solutions for the major of the instances. Note however, the Evolutionary Algorithm solutions are better than Tabu Search, for the smallest instances. For the biggest ones occurs the opposite. Finally the analyzed strategies improve the logistic enterprise solutions for the real problem.

In any case, this paper is a first approach, because of the analyzed strategies are basic. In future works we hope improve this strategies using another elements such as short-term memory, long-term memory and solutions set update methods. Also, we'll analyze another strategies and other types of neighborhood moves.