ESTUDIOS DE ECONOMÍA APLICADA VOL. 21 - 3, 2003. PÁGS. 537-557

Estudio comparativo de diferentes estrategias metaheurísticas para la resolución del labor scheduling problem

CASADO YUSTA, S. y PACHECO BONROSTRO, J.
Facultad de CC.EE. y EE. Universidad de Burgos.
Plza. Intanta elena, s/n. 09001 Burgos. Telf.; 947 259 024. Fax: 947 258 956. E-mail: scasado@ubu.es

RESUMEN

En este trabajo se analiza el labor scheduling problem que consiste en programar el horario de trabajo o los turnos de los distintos empleados de forma que se minimicen los gastos de personal y los costes de oportunidad esperados. Dado que generalmente las empresas y organismos mantienen un elevado nivel de flexibilidad en sus horarios, tradicionalmente muchos procedimientos usados para la resolución de problemas de turnos de horarios han sido estrategias heurísticas clásicas. Sin embargo son pocos los trabajos existentes en la literatura que propongan estrategias metaheurísticas, que son de más reciente creación. Además estos trabajos solo usan un pequeño número de estas estrategias, -*Busqueda Tabu, Temple Simulado y Algoritmos Genéticos*-, no existiendo referencias destacadas que usen otros Metaheurísticos. La aportación de este trabajo ha sido el diseño y posterior estudio comparativo de una amplia gama de metaheurísticos para este problema, además de un eficaz tipo de movimientos vecinales. Concretamente, las estrategias que se usan en este trabajo son GRASP, *Búsqueda en Entornos Variables, Temple Simulado, Búsqueda Tabú, Algoritmos Genéticos y Algoritmos Meméticos*. Para realizar esta comparación se hacen pruebas con diferentes instancias ficticias con un horizonte temporal de una semana.

Palabras clave: labor scheduling, movimientos vecinales, estrategias metaheurísticas.

Analisys of different methauristas for solving labor scheduling

ABSTRACT

In this work, the labor-scheduling problem is analyzed. This problem consists of planning the schedule or the shifts for the employees minimizing labor and opportunity costs. Because of the firms, generally, have a high degree of scheduling flexibility, heuristics methods have been used to solve this problem. However there are few works in the literature that propose metaheuristics strategies for this problem, and besides only a small number of this strategies are used, -*Tabu Search, Simulated Annealing and Genetic Algorithms*-. There are not important references that use another metaheuristics procedures. The contribution of this work is the design and comparison of a wide range of metaheuristics for this problem and also an effective type of neighborhood is designed. Specifically, GRASP, *Variable Neighborhood Search, Simulated Annealing, Tabu Search, Genetic Algorithm and Memetic Algorithm.* Computational experiences with different fictitious instances for a planing horizon of a week are performed.

Key words: Labor Scheduling, local movements, metaheuristics strategies.

Clasificación JEL C61

Artículo recibido en octubre de 2002. Aceptado en octubre de 2003.

1. INTRODUCCIÓN

Una decisión que se requiere en la organización y funcionamiento de muchas empresas de servicios es determinar el número de empleados y sus horarios de trabajo de forma que se minimicen los gastos de personal y los costes de oportunidad esperados. Tradicionalmente a este problema se le denomina *labor scheduling problem.*

> *"Labor scheduling -el proceso que iguala, durante las operaciones del día, el número de empleados trabajando con el número de trabajadores que se necesitan para proveer el nivel deseado de servicio al cliente- es frecuentemente un gran determinante de la eficiencia en la organización del servicio"*. (Thompson, 1995).

El desarrollo de horarios de trabajo eficientes ha sido reconocido durante un largo tiempo como un factor elemental para mejorar la productividad en los servicios. Muchas empresas de servicios tienen demandas que varían significativamente de una hora a otra y de un día a otro. Si esas empresas carecen de capacidad para satisfacer la demanda cuando esta se presenta pueden incurrir en pérdidas o aumentar sus costes (imagen, pérdida de clientes). De igual forma, un exceso de la capacidad de oferta sobre la demanda podría conducir al despilfarro.

Dantzig (1954) fue el primero en formular el problema de *labor scheduling* como un modelo matemático. La formulación propuesta por él es la siguiente:

$$Min \sum_{j=1}^{m} c_j x_j$$

sujeto a

$$\sum_{j=1}^{m} a_{j,i} x_j \geq r_i \; ; \; \text{i=1..h}$$

$$x_j \geq 0 \; ; \quad x_j \in Z \; ;$$

donde

h = número de periodos de tiempo considerados (normalmente horas).
m = número de turnos posibles o permitidos.

$$a_{j,i} = \begin{cases} 1, \text{ si el periodo i está incluido en el turno j,} \\ \\ 0, \text{ en otro caso} \end{cases} \qquad i=1..h \text{ y } j=1..m.$$

c_j = coste de tener un empleado trabajando durante el turno *j, j=1..m,*
r_i = nivel requerido de trabajadores en el periodo *i, i=1..h,*
x_j = número de empleados trabajando en el turno *j, j=1..m.*

El objetivo de este modelo es minimizar el coste de los turnos programados en el horizonte temporal considerado (ya sea un día o una semana) sujeto a que haya suficiente número de trabajadores en todos los periodos para satisfacer la demanda.

Otros modelos de *labor scheduling* son los de Moondra (1976), Baker (1976), Keith (1979), Betchold y Jacobs (1990) y (1991), Betchold y otros (1991), y Thompson (1990), (1992), (1995) y (1997).

Dado que generalmente las empresas (públicas o privadas) mantienen un elevado nivel de flexibilidad en sus horarios, el número de turnos disponibles suele ser muy elevado. Además Bartholdi (1981) mostró que estos problemas son NP-completos. Es decir se requiere un tiempo elevadísimo para obtener la solución óptima incluso en problemas de pequeño tamaño. Esto motiva el desarrollo de los convencionales métodos de solución heurísticos y metaheurísticos, (más rápidos que los métodos óptimos y que dan lugar a soluciones seudoóptimas). Varias empresas de servicios como L.L. Bean (Andrews y Parsons,1989 y 1993, y Quinn y otros, 1991), United Airlines (Holloran y Byrn, 1986) y el servicio de policía de San Francisco (Taylor y Huxley, 1989) han experimentado importantes actuaciones de mejora después de adoptar estas técnicas.

Tradicionalmente el heurístico más efectivo y convencional para este tipo de problemas combina la programación lineal con algunas formas de Búsqueda Local (*Hill Climbing*) (Bechtold et. al, 1991, Easton,1991 y Aykin,1996). Sin embargo, las técnicas de *Hill Climbing* usadas con los heurísticos más convencionales en general 'prohiben' las soluciones infactibles. Esto limita sus caminos de búsqueda a regiones factibles del espacio de búsqueda, sin la posibilidad de 'saltar' de una región factible a otra a través de regiones infactibles. Más problemática sin embargo es la tendencia de los métodos de *Hill Climbing* de converger a mínimos locales no globales. Estas limitaciones sugirieron un mayor potencial para obtener mejoras en las soluciones heurísticas. Recientemente algunos investigadores han propuesto para este trabajo diferentes técnicas metaheurísticas como el *Temple Simulado, Búsqueda Tabú, y Algoritmos Genéticos* para superar algunas de las limitaciones de los heurísticos convencionales.

Concretamente en Brusco y Jacobs (1993a y 1993b), Brusco y otros (1995) y Thompson (1996), se proponen algoritmos basados en Temple Simulado; en Easton y Mansour (1999) se diseña un Algoritmo Genético; y en Glover y McMillan (1986), Taylor y Huxley (1989), Easton y Rossin (1996), Dowsland (1998) y Alvarez-Valdes et. al (1999) se diseñan procedimientos de *Búsqueda Tabú.*

Sin embargo no hay referencias notables sobre el uso de otras estrategias metaheurísticas para este problema. Por tanto se ha creído conveniente en este trabajo desarrollar diferentes procedimientos que se basen en una mayor variedad de estrategias para comparar su eficacia en estos problemas. Concretamente se van a utilizar algunas de las estrategias más importantes y conocidas (GRASP, *Búsqueda en Entornos Variables, Temple Simulado, Búsqueda Tabú, Algoritmos Genéticos,*

Algoritmos Meméticos y Búsqueda Dispersa) para la resolución del problema del labor scheduling, comparando los resultados que se obtienen con cada una de ellas. Obsérvese como algunas de las estrategias que se utilizan en este trabajo no han sido utilizadas anteriormente para la resolución de este problema. En este caso el modelo de *labor scheduling* que se sigue es el propuesto por Dantzig (Dantzig, 1954) descrito anteriormente.

Hay que indicar que muchas de las estrategias propuestas están basadas en movimientos vecinales, e incluso incorporan procedimientos de Búsqueda Local basados en estos movimientos. Los movimientos vecinales que se proponen en este trabajo son una cadena o composición de movimientos simples. Además resultan eficaces ya que por una parte permiten visitar regiones no factibles, dando más flexibilidad a la búsqueda, a la vez que se va obteniendo en cada momento la 'proyección' de dichas soluciones en la región factible (es decir, la solución factible más cercana).

Para ello se ha diseñado el procedimiento Proyeccion similar al usado en el trabajo de Brusco y Jacobs (1993a). Sin embargo, mientras que Brusco y Jacobs solo lo usan en el último paso de cada cadena o composición de movimientos, tomando esa proyección como resultado de esa gran iteración, la aportación de nuestro trabajo es realizar la proyección a la región factible tras cada movimiento simple y quedarnos con la mejor.

El trabajo se estructura de la siguiente manera: en el apartado 2 se describen tanto los movimientos vecinales como otros procedimientos de los que hacen uso las diferentes estrategias que se describen posteriormente. En el apartado 3 se analizan estas estrategias (por este orden: GRASP, *Búsqueda en Entornos Variables, Temple Simulado, Búsqueda Tabú, Algoritmos Genéticos y Algoritmos Meméticos*). En el apartado 4 se exponen diferentes experiencias computacionales que comparan la eficacia de estas estrategias. Finalmente se muestran las conclusiones.

2. MOVIMIENTOS VECINALES

Se van a definir un tipo de movimientos vecinales que además de dar lugar a procedimientos de búsqueda local, van a ser parte fundamental de las diferentes técnicas metaheurísticas que se describirán posteriormente. Cada solución viene representada lógicamente por el vector $x = (x_1, x_2, ..., x_m)$ donde x_j representa el número de trabajadores del turno *j*.

Un simple vistazo a la función objetivo del problema indica que cualquier movimiento de mejora de una solución va a consistir en quitar trabajadores de los diferentes turnos. Esto hace que se llegue "rápidamente" a la frontera de la factibilidad. En este caso una solución sería infactible cuando el número de trabajadores contratados en una hora determinada sea inferior al que se requiere.

Se ha observado que cuando se permite "traspasar" esta frontera el proceso es más flexible y, en general, llega a mejores soluciones que cuando solo se permiten movimientos a soluciones factibles. En este último caso el procedimiento se encajona muy rápidamente.

Ahora bien, cuando se visitan soluciones infactibles es conveniente proveer al proceso de un mecanismo que permita "proyectar" dicha solución infactible a la región factible más cercana cuando se estime conveniente.

En este caso se ha diseñado un procedimiento de proyección similar al usado en el trabajo de Brusco y Jacobs (1993a). Sin embargo, las diferencias están en la forma de usarlo. Mientras que Brusco y Jacobs solo lo usan en el último paso de cada cadena de movimientos simples, tomando dicha proyección como resultado de esa iteración, en este trabajo se realiza la proyección tras cada movimiento simple tomando la mejor de ellas como resultado de esa iteración tal y como se muestra en la figura 1. Este procedimiento de proyección se describe a continuación.

Para ello se consideran los siguientes parámetros y variables:

f(x) = valor de la función objetivo para la solución *x*.

w(x) = vector de los trabajadores de que se dispone en cada periodo, correspondiente a la solución *x*.

El procedimiento Proyeccion en seudocódigo es el siguiente:

PROCEDIMIENTO PROYECCIÓN (VAR *X*)

Determinar $w_i(x) = \sum_{j=1}^{m} x_j a_{j,i}$; $i=1..h.$;

Repetir

1. $I = \{i / w_i(x) < r_i\}$, $i=1..h$;
2. *Si* $I \neq \varnothing$, *hacer*
   - 2.1. *Calcular* $necj = \sum_{i \in I} a_{j,i}$, $j=1..m$;
   - 2.2. *Calcular* $j^* / nec_j^* = max \{ necj : j=1..m \}$
   - 2.3. *Hacer:* $x_{j^*} = x_{j^*}+1$ y $w_i(x) = w_i(x) + a_{j^*,i}, i = 1...h$

hasta $I=\varnothing$

Obsérvese que el procedimiento intenta recuperar la factibilidad añadiendo el menor número de trabajadores posibles. Para ello en cada iteración elige añadir un trabajador al turno que abarque más horas infactibles. Obviamente, si la solución inicial ya es factible se la deja sin realizar ninguna variación en ella.

Los movimientos vecinales que se describen a continuación se han desarrollado también tomando ideas del trabajo de Brusco y Jacobs (1993a). Estos movimientos

se pueden considerar como una composición o cadena de movimientos simples. Cada movimiento simple consiste en quitar un trabajador de un turno.

Para elegir el turno más adecuado de donde quitar el trabajador, además del valor de la correspondiente proyección se hace uso de una función guía, *beta*. Esta función indica el número de periodos de dicho turno que permanecen factibles tras restar un trabajador a ese turno. En otras palabras, se intenta determinar con que turno el incremento en el número de las horas infactibles sería menor.

El procedimiento queda de la siguiente forma:

PROCEDIMIENTO CADENA_DE_MOVIMIENTOS (KMAX; VAR *X*):

Repetir

1. *Hacer* $x^0=x$, $k=0$, $x^m=x$ *y valor_anterior=f(x);*
2. *Mientras* $(k < kmax)$ *y* $(f(x^0) > 0)$ *hacer*

   2.1. $k=k+1$, $beta\_max = -\infty$ *y* $v2 = \infty$.

   2.2, $\forall j = 1..m$ *si* $x_j^0 > 0$ *hacer*

   2.2.1. $x^1=x^0$, $x_j^1 = x_j^1 - 1$, *calcular* $beta_j$ *y* $x^2=x^1$;

   2.2.2. *Ejecutar Proyeccion* (x^2)

   2.2.3. *Si* $(f(x^2) < v^2)$ *o* $((f(x^2) = v2)$ *y* $(beta_j > beta\_max))$ *hacer*

   2.2.3.1. $beta\_max=beta_j$, $v2 = f(x^2)$, $x''=x^2$ *y* $x^{1m}= x^1$

   2.3 *Hacer* $x^0=x^{1m}$

   2.4 *Si* $f(x'') < f(x^m)$ *hacer* $x^m=x^{2m}$

3. *Si* $f(x^m) < f(x)$ *hacer* $x = x^m$

hasta $f(x) = valor\_anterior$

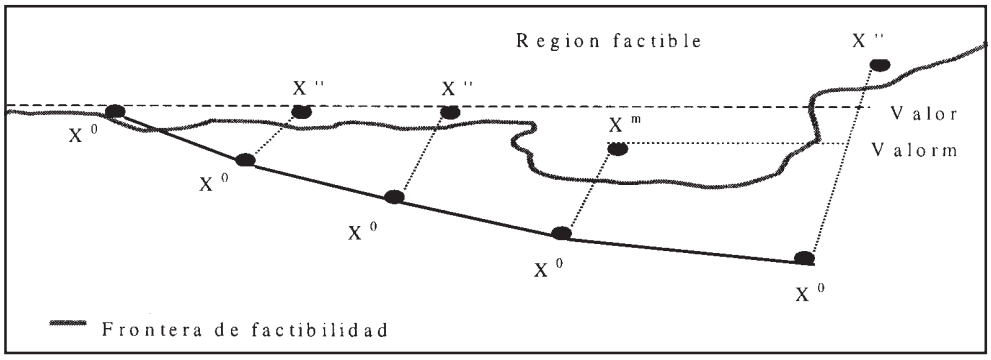
El valor de $beta_j$, $j=1\ldots m$, se calcula de la siguiente manera:

$$Cardinal \quad de \quad \left\{ i / a_{j,i} = 1 \quad y \quad w_i(x) - 1 \geq r_i \right\}$$

donde $w_i(x) = \sum_{j=1}^{m} a_{j,i} x_j^0$.

En la figura 1 se puede observar como en cada gran iteración se van formando tanto las cadenas de la solución guía x^0 como la de sus proyecciones respectivas x''. Al finalizar la iteración se toma la mejor proyección y el proceso se reinicializa hasta que no haya mejora. De esta forma, al usar simultáneamente una solución guía x^0 que puede 'recorrer' regiones infactibles y sus correspondientes proyecciones factibles, se superan algunos de los inconvenientes de los tradicionales procedimientos de *Hill Climbing* que limitaban la búsqueda a regiones factibles, según se comentó en la introducción.

Fig. 1: Formación de las cadenas de movimientos y sus proyecciones respectivas.



3. DESCRIPCIÓN DE LAS TÉCNICAS METAHEURÍSTICAS

3.1. GRASP

Las técnicas GRASP (*Greedy Randomized Adaptive Search Procedures*) fueron dadas a conocer inicialmente por Feo y Resende (1989). Dos tutoriales más recientes se pueden encontrar en Feo y Resende (1995) y Pitsoulis y Resende (2002). Básicamente actúa de la forma siguiente:

PROCEDIMIENTO GRASP (*alfa, kmax*)

Repetir
  Generar solución inicial Avido - Aleatoria x;
  Ejecutar procedimiento Cadena_de_Movimientos(kmax, x) ;
  Actualizar mejor solución;
hasta alcanzar un criterio de parada

Como método para obtener la solución inicial ávido-aleatoria en cada iteración, se emplea un algoritmo constructivo que se describe a continuación.

PROCEDIMIENTO AVIDO_ALEATORIO (*alfa*; var *x*)

Iniciar : $x_j=0, j=1..m$ *y* $w_i(x)=0, i=1..h;$
Repetir
  1. $I = \{i / w_i(x) < r_i\}$;
  2. Si I Ø *hacer*
    2.1 Calcular $nec_j = \sum_{i \in I} a_{j,i}$;

2.2. Determinar: $nec\_max = max\ \{nec_j ; j = 1..m\}$ *y*

$nec\_min = min\ \{nec_j ; j = 1..m\}$

2.3. Formar $L = \{j / nec_j \geq alfa * nec\_max + (1 - alfa) * nec\_min\}$;

2.4. Elegir aleatoriamente $j^* \in L$;

2.5 Hacer: $x_{j^*} = x_{j^*} + 1$ *y* $w_i(x) = w_i(x) + a_{j^*,i}, i = 1...h$

hasta I=Ø

En este caso el procedimiento usa como función guía *nec(j)* que indica para cada turno j a cuantas horas infactibles de las que abarca ayudaría a recuperar la factibilidad si se añade un trabajador a dicho turno. Es por tanto una medida de la "bondad" de cada turno para elegir el más adecuado para añadir trabajadores.

Obsérvese que no necesariamente se elige el "mejor" (el de mayor valor de nec) sino se selecciona aleatoriamente uno de los mejores (los que forman la lista L de mejores candidatos). Se ha demostrado como con este "equilibrio" entre aleatoriedad y agresividad (o avidez) se consiguen mejores resultados que usando sólo estrategias totalmente ávidas o totalmente aleatorias.

El parámetro que mide el grado de aleatoriedad o avidez es *alfa*. Para *alfa=0*, $L = \{j / j = 1..m\}$, es decir, estaría formado por todos los *j* y el procedimiento sería totalmente aleatorio. Por el contrario si *alfa=1,* la lista *L* solamente estaría formada por el mejor (o los mejores en caso de que haya más de uno que alcance el máximo) y el procedimiento sería totalmente ávido y mucho más determinístico que con valores más bajos. Hay que señalar que, en este caso, con *alfa=1* el procedimiento constructivo Avido_Aleatorio no es totalmente determinístico. Esto se debe a que en cada iteración del método constructivo, se dan en muchas ocasiones 'empates'. Es decir, hay varios turnos j con el máximo valor de la función *nec*. Por tanto la lista de candidatos *L* puede estar formada por más de un elemento, lo que hace que, incluso para este valor de *alfa*, exista un cierto grado importante de aleatoriedad en el proceso.

3.2 Búsqueda en entornos variables

La *Búsqueda en Entorno Variable* o VNS (*Variable Neighborhood Search*) es un reciente metaheurístico para resolver problemas combinatorios propuesto y descrito en trabajos como Mladenovic (1995), Mladenovic y Hansen (1997) y Hansen y Mladenovic (1998). Un tutorial más reciente se puede encontrar en Hansen y Mladenovic (2002).

En nuestro caso el algoritmo VNS actúa de la forma siguiente:

PROCEDIMIENTO VNS (*pmax, maxiter, kmax;* var *x*):

Hacer: niter=0 e iter_mejor=0;
Repetir
1. Hacer k=0 y niter=niter+1;
2. Repetir
2.1 Hacer: k=k+1 y $x^1=x$;
2.2 Ejecutar Perturbar *(k,*x^1*);*
2.3 Ejecutar Proyección *(*x^1*);*
2.4Ejecutar Quitar *(x1);*
2.5 Ejecutar Cadena_de_Movimientos *(*x^1*, kmax);*
hasta (k = pmax) o ($f(x^1) < f(x)$*) (fin paso 2)*
3. Si $f(x^1) < f(x)$ *hacer:* $x=x^1$ *e iter_mejor=k;*
hasta (niter $\geq$ *iter_mejor+maxiter) u otro criterio de parada;*

El procedimiento Perturbar consiste en tomar k turnos y asignarlos un valor entero de forma aleatoria. El rango de valores que pueden tomar varía desde 0 hasta un máximo específico para cada turno j. En este caso, este máximo se define como

$$max\_trabj = max\{r_i / a_{j,i} = 1, i = 1..h\}, j = 1..m .$$

Obviamente es innecesario asignar a un turno más trabajadores de los que se necesitan en la hora en la que más trabajadores se requieren, entre las abarcadas por dicho turno. Según esto, el procedimiento Perturbar en seudocódigo es como sigue:

PROCEDIMIENTO PERTURBAR (*k*; var *x*)

Seleccionar aleatoriamente k turnos $j_1, j_2,\ldots,j_k$;
Para l=1..k, asignar a x_{j1} *un valor entero aleatorio entre 0 y* max_trab_{j1};

Obsérvese que esta "perturbación" no garantiza ni la factibilidad de la solución obtenida ni que esta sea de calidad. Por eso se requieren procedimientos que la devuelvan la factibilidad y/o que mejoren su calidad de forma rápida. Para lo primero se aplica el procedimiento Proyeccion explicado anteriormente. Una vez conseguida la factibilidad, con el procedimiento Quitar se pueden obtener mejoras y conseguir una solución de calidad aceptable.

El procedimiento Quitar actúa de la forma siguiente:

PROCEDIMIENTO QUITAR (var *x*)

Determinar $w_i(x) = \sum_{j=1}^{m} x_j a_{j,i}$ *; 1=1..h.*

Repetir

$\forall j, j = 1...m$ *hacer: calcular* $H_j = \min\{w_i(x) - r_i / \forall i / a_{j,i} = 1\}$;

si $H_j > x_j$ *hacer* $H_j = x_j$;

Determinar $H_{j*} = \max\{H_j, j = 1..m\}$;

Si $H_{j*} > 0$ *hacer:*

$x_{j*} = x_{j*} - H_{j*}$; $w_i(x) = w_i(x) - a_{j,i}, i = 1...h$;

hasta $H_j = 0$;

El procedimiento Quitar asegura la obtención de buenas soluciones que posteriormente son mejoradas por el procedimiento Cadena_de_Movimientos. Hay que señalar que los resultados obtenidos por este procedimiento de mejora así como su tiempo de computación dependen de la calidad de la solución de partida. Si esta es mala, los resultados finales son peores y con mayor tiempo de computación que si se parte de soluciones iniciales de una calidad aceptable.

3.3. Temple simulado

Este metaheurístico fue propuesto por Kirkpatrick y otros (1983) y Cerny (1985). En él se permiten movimientos hacia soluciones peores aunque de forma controlada.

El esquema de nuestro algoritmo de *Temple Simulado* en seudocódigo es el siguiente:

PROCEDIMIENTO TEMPLE_SIMULADO (*x, pmax, t_inicial, c;* var *xm*)

1. H*acer : temp = t_inicial y xm = x ;*
2. *Repetir*
   2.1. Ejecutar Generar_Vecino (x, x^0*);*
   2.2. Generar aleatoriamente s uniformemente en (0,1);
   2.3. Si (f(x^0*) f(x)) o (s < exp(f(x)-f(*x^0*)/temp)) hacer x=*x^0*;*
   2.4. Si f(x) < f(x^m*) hacer* x^m*=x;*
   *2.5. Hacer temp= temp*c;*

hasta alcanzar condición de parada.

El procedimiento Generar_Vecino es el siguiente:

PROCEDIMIENTO GENERAR_VECINO (x; var x0);

1. *Hacer* $x^0 = x$;
2. *Ejecutar* Perturbar (*pmax,* x^0);
3. *Ejecutar* Proyeccion (x^0);
4. *Ejecutar* Quitar (x^0);

Para determinar la temperatura inicial (*t_inicial*) se ha usado un procedimiento que obtiene un valor de esta con la que cualquier cambio en el entorno de la solución inicial es admitido con una probabilidad de al menos 0.5; Para ello se considera 'el peor de los casos' (es decir, el movimiento 'más difícil' o menos probable de ser aceptado): el paso de la mejor a la peor solución en el entorno de la solución inicial. A continuación se calcula que temperatura hace falta para que ese cambio sea admitido con una probabilidad de al menos 0.5. El procedimiento en seudocódigo queda como sigue:

PROCEDIMIENTO TEMPERATURA_INICIAL (*n_vecinos, pmax, x*):

1. *Hacer: min = infinito y max = 0;*
2. $\forall i = 1..n\_vecinos$ *hacer:*
   2.1. Ejecutar Generar_Vecino *(x, x^0);*
   2.2. Si f(x) > max hacer max = f(x);
   2.3. Si f(x) < min hacer min = f(x);
3. *Hacer t_inicial = (max-min) / ln2;*

Finalmente, el algoritmo de Temple Simulado se parará cuando la temperatura alcance un valor tan bajo que la probabilidad de que se de cualquier movimiento a peor o hacia arriba sea insignificante (0.001), y por tanto sea casi imposible salir del 'valle' del mínimo local actual.

3.4 Búsqueda Tabú

Los primeros trabajos sobre *Búsqueda Tabú* con tal nombre se deben a Glover (1989) y (1990), aunque los principios básicos ya se explicaron en un trabajo anterior del mismo autor en 1986, (Glover (1986)). Esta estrategia 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)). Se permiten cambios hacia arriba o que empeoran la solución, una vez que se llega a un óptimo local. Simultáneamente los últimos movimientos se califican como tabús durante las siguientes iteraciones para evitar que se vuelva a soluciones anteriores y el algoritmo cicle. El termino tabú hace referencia a *"un tipo de inhibición a algo debido a connotaciones culturales o históricas y que puede ser superada en determinadas condiciones..."*. (Glover, 1996). Recientes y amplios tutoriales sobre Búsqueda Tabú, que incluyen todo tipo de aplicaciones, pueden encontrarse en Glover y Laguna (1997) y (2002).

El esquema de funcionamiento de este algoritmo en seudocódigo es el siguiente:

PROCEDIMIENTO BUSQUEDA_TABU (x, var x^m):

Hacer $x^m = x$ *y niter=0*
Repetir
0. Hacer k=0 , $x^0 = x$ *y niter=niter+1:*
1. Repetir
1.1. Hacer k=k+1;

1.2. Definir N= $\left\{ j / j \text{ no es tabú o cumple criterio de aspiración y } x_j^0 > 0 \right\}$

1.3. Si N Ø hacer

- Determinar $beta_{j^*} = max \left\{ beta_j / j \in N \right\}$

- Hacer = -1 y , i=1..h

- Hacer x'=x^0 *y ejecutar Proyección(x')*

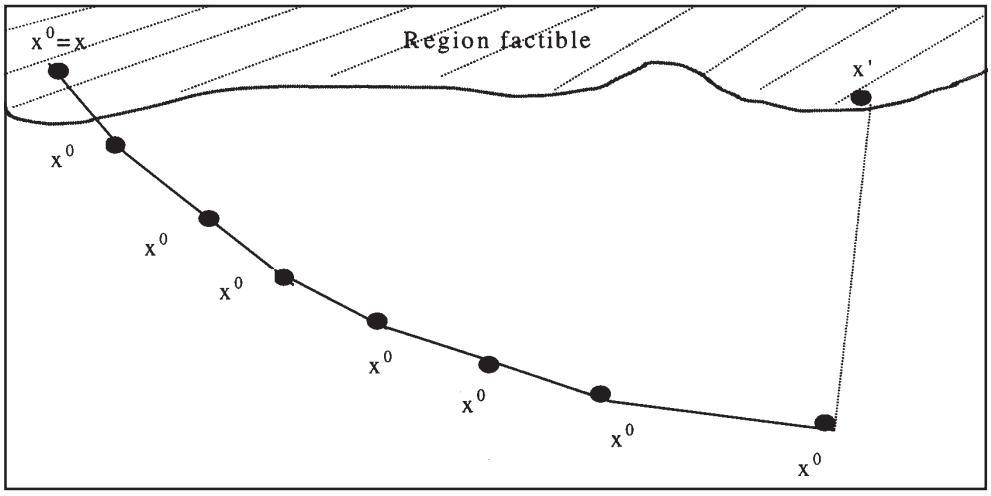
hasta (k=ka) o (N =Ø) o ($f(x') < f(xm)$ *) {Fin paso 1}*
2. Definir A = {$j / x'_j > x_j$*} y Hacer lista_tabúj=niter,* $j \in A$ *{Actualizar}*
3. Hacer x=x' y actualizar x^m *{ Mejor Solución encontrada }*
Hasta criterio de parada.

Como se puede observar este procedimiento sigue una estrategia similar al de Cadena_de_Movimientos: alejarse de la región factible quitando elementos y volver a ella a continuación con el procedimiento Proyeccion. Esta idea de entrar y salir de la región factible secuencialmente aparece también en el trabajo de Alvarez-Valdes y otros, (1999).

En nuestro caso, en cada gran iteración se van eliminando elementos de los diferentes turnos *j* siguiendo la función beta definida en apartados anteriores (*betaj* = número de horas que permanecerían factibles, entre las abarcadas por *j*, si se quitara un trabajador a dicho turno). Cada gran iteración finaliza cuando se ejecutan un número de extracciones prefijado (*ka*) o cuando la correspondiente proyección a la región factible mejore la mejor solución encontrada hasta ese momento. Este modo de funcionar se ilustra en la figura 2.

Sean x y x' las soluciones de partida y llegada respectivamente en cada una de estas iteraciones, se determina el conjunto A= $\left\{ j / x_j < x'_j \right\}$, es decir el conjunto de turnos donde se han añadido recursos. Para evitar que el algoritmo cicle en las siguientes iteraciones (*tabu_tenure*) se va a impedir que se extraigan trabajadores de los elementos o turnos de A (movimientos tabú). Para ello se hará uso de un vector tabú (*lista_tabu*) que indica para cada turno *j* en que iteración se han añadido trabajadores. Esta condición tabú puede ser ignorada cuando dicho movimiento diera lugar

Fig. 2: Procedimiento Búsqueda_Tabú



a una solución que sigue siendo factible y además mejor que la mejor solución encontrada anteriormente (*criterio de aspiración*). El algoritmo termina cuando transcurren un número determinado de iteraciones sin mejora u otro criterio de parada.

3.5. Algoritmos genéticos

Según Goldberg, (1989) *"Los Algoritmos Genéticos son técnicas de búsqueda basadas en la mecánica de la selección natural y la genética"*. Estas técnicas son probablemente el representante más conocido de los algoritmos evolutivos, y aquellas cuyo uso está más extendido. Fueron concebidas originalmente por John Holland y descritas en el ya clásico *"Adaptation in Natura and Artificial Systems"* (Holland, 1975). Este texto ha tenido una gran transcendencia en el posterior desarrollo de estas técnicas ya que los mecanismos en él descritos han sido tomados durante largo tiempo como auténticos dogmas.

Están basados en una estrecha analogía con los procesos de evolución natural de las especies. La base es un conjunto (población) de soluciones sobre las que se realizan operaciones tales como Selección, Cruce, Reproducción, Renovación o Mutación. Su característica principal es el uso del operador de recombinación o cruce como mecanismo principal de búsqueda: construye descendientes que poseen características de los cromosomas que se cruzan. Su utilidad viene dada por la suposición de que diferentes partes de la solución óptima pueden ser descubiertas independientemente y luego ser combinadas para formar mejores soluciones.

Un esquema de funcionamiento de un Algoritmo Genético puede ser el siguiente:

PROCEDIMIENTO ALGORITMO_GENÉTICO (*pmut*, *npob*, *nsel;* var *x*):

Generar una población inicial de soluciones aletorias de tamaño npob.
Repetir
- *Seleccionar aleatoriamente un subconjunto de nsel elementos (par) de la población con una probabilidad proporcional a su bondad.*
- *Cruce Reproducción: Emparejar o cruzar estas soluciones (Padres) para dar lugar a nuevos soluciones (Hijos). De cada pareja de Padres se han den generar una nueva pareja de hijos.*
- *Sustituir las peores soluciones de la población por las nuevas soluciones hijas.*
- *Mutación: Los soluciones de la población pueden cambiar con una probabilidad pmut alguno de sus elementos (genes).*

hasta conseguir algún criterio de parada.
Hacer x la mejor de las soluciones de la población final.

Al igual que en apartados anteriores la solución va a venir representada por un vector $x = (x_1, x_2, ..x_j, ..., x_m)$, donde x_j es el número de trabajadores contratados en el turno *j*.

La no-bondad de cada solución de la población va a venir dada por el valor de la función objetivo *f* de la solución correspondiente. La probabilidad de selección de una determinada solución x va a ser proporcional a *fmax - f(x)*; donde *fmax* es el máximo valor de la función objetivo en la población en cada iteración.

Las operaciones de cruce y mutación se realizan sobre los vectores x de cada solución. El operador de cruce sigue el esquema 'one-point crossing', es decir, sean x y x' dos vectores padres seleccionados:

$$x = (x_1, x_2, x_3, ...x_m) \quad \text{y} \quad x' = (x'_1, x'_2, x'_3, ...x'_m)$$

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

$$x''' = (x'_1, x'_2, ..., x'_{pto\_cruce}, x_{pto\_cruce+1}, x_m)$$

Cada componente *j-ésima* (turno *j*) de cada nueva solución hija puede cambiar o mutar con una 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 número entre 0 y max_trab_j (definido en apartados anteriores) y asignárselo a dicho turno. 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 generado un número (par) de soluciones hijas (tantas como padres se hallan seleccionado) si alguna de ellas es mejor que la peor de la población actual la reemplaza, de forma que el número de elementos permanece constante (*Renovación*).

Cada una de las iteraciones en las que se realiza la secuencia de *Selección* de Padres, *Cruce o Reproducción, Mutación y Renovación* se denomina generación. El algoritmo termina cuando transcurren un determinado número de generaciones sin mejora, o cuando se cumplen otros criterios predeterminados.

3.6. Algoritmos meméticos.

Los Algoritmos Meméticos también son procedimientos basados en población y se han mostrado como técnicas más rápidas que los tradicionales *Algoritmos Genéticos* para algunos tipos de problemas. Básicamente combinan procedimientos de Búsqueda Local con operadores de cruce o mutación, por lo que algunos investigadores les han visto como *Algoritmos Genéticos Híbridos* o *Algoritmos Genéticos Paralelos* (PGAs) o métodos de *Búsqueda Local Genética.*

El método, descrito originalmente en el trabajo de Moscato (1989) está ganando amplia aceptación, en particular en los bien conocidos problemas de optimización combinatoria donde grandes ejemplos se han solventado óptimamente y donde otras estrategias metaheurísticas han fallado. Dos recientes tutoriales sobre Algoritmos Meméticos se puede encontrar en Moscato, (2002), y Moscato y Cotta, (2002). Así mismo información sobre Algoritmos Meméticos se puede encontrar en Memetic Algorithms' Home Page: http://densis.fee.unicamp.br/~moscato/memetic_home.html

En nuestro caso el Algoritmo Memético que se propone sigue el mismo funcionamiento que el Algoritmo Genético descrito en el apartado anterior salvo que se añaden la ejecución del procedimiento Cadena_de_Movimientos (con *kmax* =5) cada vez que se genera una nueva solución: concretamente tras la generación de cada solución inicial y tras la ejecución de la *Mutación*.

4. EXPERIENCIAS COMPUTACIONALES

Para la realización de las pruebas se utilizan dos conjuntos de 49 instancias ficticias cada uno. En el primero el número de turnos es de 123 mientras que en el segundo es de 73. El horizonte temporal considerado es de una semana dividido en periodos de tiempo de 1 hora, por lo hay un total de 168 horas. Dentro de cada conjunto de instancias los turnos son los mismos, cambiando únicamente el vector de recursos requeridos (*r*).

Los valores de los parámetros usados en la implementación de las distintas estrategias se exponen a continuación (se ha seleccionado el valor de los parámetros más adecuado según estudios realizados previamente):

- GRASP: *maxiter* =100, *alfa* =1 y *kmax* =5
- *Búsqueda en Entornos Variables* (VNS): *pmax* = 8, *maxiter* = 10 y *kmax* = 5
- *Temple Simulado,* (TS): *pmax*= 4 y *c*=0.999.

- *Algoritmo Genético,* (AG): *npob* =50, *nse*l =5, y *pmut* =0.05 (500 generaciones como criterio de parada)
- *Algoritmo Memético*, (AM): *npob* =20, *nse*l =2, y *pmut* =0.033 (8 generaciones como criterio de parada)
- *Búsqueda Tabú,* (BT): *ka*=20, *tabu_tenure*=10(m/12) y *maxiter*=2000

El resumen de los resultados obtenidos con el primer conjunto de instancias (*m*=123) se exponen en la tabla 1, que incluye para cada estrategia el valor agregado obtenido en el conjunto de las 49 instancias, la suma de los tiempos totales de cálculo (segundos) y la suma de los tiempos de cálculo (segundos) hasta alcanzar la mejor solución encontrada durante el proceso.

Tabla 1: Comparación de las distintas estrategias metaheurísticas cuando *m*=123.

	GRASP	VNS	TS	AG	AM	BT
Valor	3291	3319	3311	3290	3281	3274
t.total.	828,07	207,03	279,43	1702,00	13711,77	960,24
t.mej.sol.	159,57	81,32	161,63	588,31	8606,87	148,87

De la tabla 1 se obtienen las siguientes consecuencias:

- Se observa como la estrategia que nos permiten obtener los mejores resultados en un tiempo de computación corto es *Búsqueda Tabú.* También dan buenos resultados el *Algoritmo Memético* y en menor medida el *Algoritmo Genético* y GRASP. En cuanto a *Temple Simulado* y *Búsqueda en Entornos Variables*, estos obtienen buenos resultados rápidamente, pero no muestran capacidad de mejorarlos posteriormente.
- Los *Algoritmos Genético* y *Memético* dan soluciones algo peores que *Búsqueda Tabú*, además emplean un tiempo de computación mucho mayor que el resto de las estrategias para alcanzar estas soluciones. Llama la atención especialmente el tiempo de computación empleado por el *Algoritmo Memético* (8606,87 segundos en alcanzar la mejor solución). En este punto creemos necesario hacer el siguiente comentario: Se ha observado como prácticamente el 80% del tiempo de computación que usa este algoritmo lo emplea en generar y mejorar las soluciones iniciales. Al partir de soluciones iniciales totalmente aleatorias, que resultan ser de mala calidad, el procedimiento de mejora de dichas soluciones, Cadena_de_Movimientos, emplean un excesivo tiempo de computación. Por tanto una recomendación clara que se puede obtener es el uso de procedimientos que tengan una componente aleatoria pero controlada y sobre todo que utilicen criterios que incorporen información del problema (como el procedimiento Avido_Aleatorio).
- La estrategia GRASP, que precisamente usa de estos procedimientos. Ávido-Aleatorios para generar soluciones iniciales de calidad, llega a soluciones fina-

les parecidas a las obtenidas por los *Algoritmos Genético* y *Memético* en mucho menor tiempo que estas dos estrategias.

Se han repetido estas pruebas con el segundo conjunto de 49 instancias. En este caso el número de turnos es de 72 y el número de horas sigue siendo 168. En la tabla 2 se resumen como en el caso anterior los resultados obtenidos.

Tabla 2: Comparación de las distintas estrategias metaheurísticas cuando m=72

	GRASP	VNS	TS	AG	AM	BT
Valor	3302	3310	3301	3306	3291	3289
t.total.	476.84	121.48	193.51	1083.13	4535.87	644.56
t.mej.sol.	104.22	41.22	110.48	298.09	1713.12	82.82

Las consecuencias que se obtienen de los resultados de la tabla 2 son muy similares a los de la tabla 1:

- La mejor estrategia globalmente sigue siendo *Búsqueda Tabú*: obtiene las mejores soluciones y el tiempo empleado hasta alcanzarlas solo es rebajado por *Búsqueda en Entornos Variables* que llega a resultados claramente peores.
- En un segundo plano, en cuanto a las soluciones obtenidas destaca el *Algoritmo Memético*. Sin embargo sigue llamando la atención el excesivo tiempo empleado por éste, (por las razones ya comentadas) para alcanzar estas soluciones frente a la rapidez de la *Búsqueda Tabú* (menos del 5% de tiempo del empleado por el *Algoritmo Memético*).
- GRASP, *Temple Simulado* y el *Algoritmo Genético* obtienen soluciones de calidad parecida. Si bien este último emplea un tiempo de cálculo significativamente mayor que las otras 2 para alcanzar estas soluciones.
- Al igual que en las pruebas anteriores *Búsqueda en Entornos Variables* obtiene sus mejores soluciones muy rápidamente pero no muestra 'capacidad de mejora' obteniendo los peores resultados.

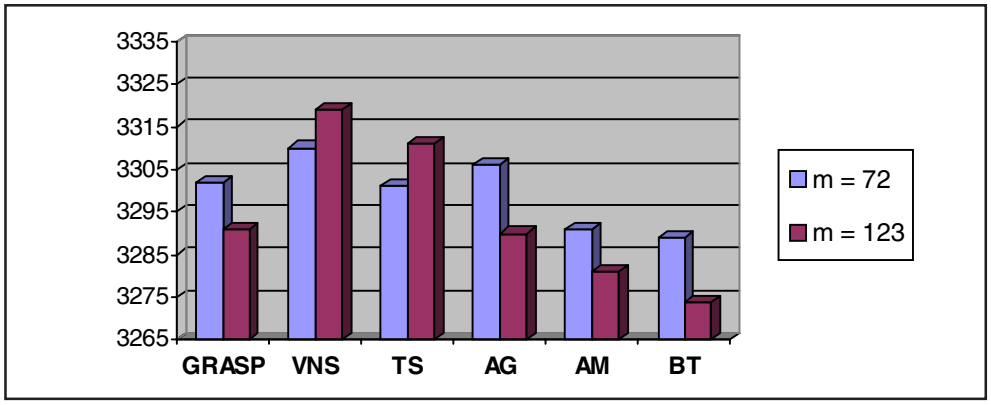
Finalmente de las comparaciones de estas 2 tablas se ha creído conveniente añadir los siguientes comentarios:

- Como era de esperar el tiempo computación aumenta con el número de turnos disponibles (72 frente a 123), ya que la complejidad es mayor. Hay que indicar que ambos conjuntos de instancias ficticias (instancia a instancia) usan los mismos vectores r de recursos requeridos y lo que cambia únicamente es el número de turnos disponible.
- Llama la atención el excesivo aumento en el tiempo del Algoritmo Memético (5 veces más). Esto es debido, relacionado con lo comentado anteriormente, a que con la forma de generar soluciones aleatorias en esta estrategia, cuanto más turnos hay (componentes de cada solución x), más valor tienen y peores son.

- Exceptuando Búsqueda en Entornos Variables y Temple Simulado, se observa como estrategia a estrategia mejoran los resultados según aumenta m. Esto se debe a que con más turnos aunque la complejidad aumenta también son mayores las posibilidades de búsqueda.

La figura 3 ilustra este comportamiento.

Fig. 3.Comparación de resultados para distintos valores de m.



5. CONCLUSIONES

La planificación eficiente de horarios de trabajo es un factor elemental a la hora de mejorar la productividad en las empresas de servicios. En este trabajo se analiza el problema de asignar un número de empleados a cada horario de trabajo de forma que se minimicen los costes del personal. Este problema es conocido tradicionalmente como problema del *labor scheduling*. Este es un problema NP-hard y normalmente de gran tamaño por lo que tradicionalmente se han propuesto para su resolución métodos heurísticos tradicionales. Sin embargo, son pocos las referencias en la literatura donde se empleen técnicas metaheurísticas más avanzadas para este problema que superen algunos de los inconvenientes de los tradicionales heurísticos. Por ello en este trabajo se diseñan y analizan diferentes procedimientos basados en las principales estrategias metaheurísticas tales como GRASP, *Búsqueda en Entornos Variables, Temple Simulado, Búsqueda Tabú, Algoritmos Genéticos* y *Algoritmos Meméticos.* De la comparación realizada entre estos métodos, con problemas de diferente tamaño, destaca el basado en *Búsqueda Tabú*, tanto en calidad de la solución como en rapidez para obtenerla. También se propone un efectivo tipo movimientos vecinales entre soluciones del que hacen uso casi todas las estrategias anteriores.

BIBLIOGRAFIA

ALVAREZ-VALDES R., CRESPO E. and TAMARIT J.M. (1999). Labor Scheduling at an Airport Refuelling Installation. Journal of the Operational Research Society, vol. 50, nº 3, pp. 211-218.

ANDREWS B. and PARSONS H. (1989). L.L. Bean Chooses a Telephone Agent Scheduling System. Interfaces, Vol. 19, nº 6, pp. 1-9.

ANDREWS B. and PARSONS H. (1993). Establishing Telephone-Agent Staffing Levels Through Economic Optimization. Interfaces, Vol. 23, nº 2, pp. 14-20.

AYKIN T.(1996). Optimal Shift Scheduling with Multiple Break Window. Management Science, Vol. 42, nº 4, pp. 591-602.

BAKER K.R. (1976). Workforce Allocation in Cyclical Scheduling Problems: A survey. Operational Research Quarterly, 27, pp. 155-167.

BARTHOLDI, J. J. (1981). A Guaranteed-Accuracy Round-off Algorithm for Cyclic Scheduling and Set Covering. Operations Research , Vol. 29, nº 3. Pp.501-510.

BECHTOLD S.E. and JACOBS L.W.(1990). Implicit optimal modeling of flexible break assignments in labor staffing decisions for service operations. Management Science , 36, 11, pp. 1339-1351.

BECHTOLD S.E. and JACOBS L.W.(1991). Improvement of labor utilization in shift scheduling for services with implicit optimal modeling. International J. of Oper. An Production Management, 11, 2, pp. 54-69.

BETCHOLD S.E.M BRUSCO M. and SHOWALTER M. (1991). A Comparative Evaluation of Labor Tour Scheduling Methods. Decision Science, 22, pp.683-699.

BRUSCO M. and JACOBS L. (1993a). A Simulated Annealing Approach to the Cyclic Staff-Scheduling Problem. Naval Research Logistics, 40(1), pp. 69-84.

BRUSCO M. and JACOBS L. (1993b). A Simulated Annealing Approach to the Solution of Flexible Labour Scheduling Problems. Journal of the Operational Research Society, Vol 44, nº 12, pp. 1191-1200.

BRUSCO M., JACOBS L., BONGIORNO R., LYONS D. and TANG B (1995). Improving Personnel Scheduling at Airline Stations. Operations Research 43 (5), pp.741-751.

CERNY V. (1985). Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm. Jounal of Optimization Theory and Applications, vol.45,pp. 41-51.

DANTZIG G.B. (1954). A Comment on Edie´s `Traffic Delays at Toll Booths´. Operations Research, 2, 3, pp. 339-341.

DOWNSLAND K.A. (1998). Nurse Scheduling with Tabu Search and Strategic Oscillation. European Journal of Operational Research, 106, pp. 393-407.

EASTON F. (1991). Modular IP Aproximation Procedures for Tour Scheduling Problems. Proceedings of the National Conference, Decision Sciences Institute, Atlanta, GA.

EASTON F. and MANSOUR N. (1999). A Distributed Genetic Algorithm for Employee Staffing and Scheduling Problems. In : S. Forrest (Ed.) , Proceedings of Fifth International Conference on Genetic Algorithms, Morgan-Kaufman, San Mateo, CA, pp. 360-367.

EASTON F. and ROSSIN D. (1996). A Stochastic Goal Program for Employee Scheduling. Decision Sciences 27 (3), pp.541-568.)

FEO T.A. and RESENDE M.G.C.(1989). 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.(1995). Greedy Randomized Adaptative Search Procedures. Journal of Gloval Optimization. Vol. 2. Pp. 1-27.

GLOVER F. (1986). Future Paths for Integer Programming and Links to Artificial Intelligence. Computers and Operations Research, Vol. 13, pp. 533-549.

GLOVER F. (1989). Tabu Search: Part I. ORSA Journal on Computing. Vol. 1, pp. 190?206.

GLOVER F. (1990). Tabu Search: Part II. ORSA Journal on Computing. Vol. 2, pp. 4?32.

GLOVER F.(1996). Búsqueda Tabú en Optimización Heurística y Redes Neuronales. Adenso Díaz (coordinador). Paraninfo. Madrid, pp. 105-143.

GLOVER F. and LAGUNA M. (1993). Tabu Search in Modern Heuristic Techniques for Combinatorial Problems. C. Reeves ed., Blackwell Scientific Publishing, pp. 70?141.

GLOVER F. and LAGUNA M. (1997). Tabu Search. Kluwer Academic Publishers, Boston.

GLOVER, F. and LAGUNA, M. (2002). Tabu Search, in Handbook of Applied Optimization, P. M. Pardalos and M. G. C. Resende (Eds.), Oxford University Press, pp. 194-208.

GLOVER F. and McMILLAN C.(1986). The General Employee Scheduling Problem: an Integration of Management Science and Artificial Intelligence. Computers and Operations Research 13 (5), pp. 563-593.

GOLDBERG D. (1989). Genetic Algorithms in Search, Optimization and Machine Learning. Addision-Wesley, reading, MA.

HANSEN P. and MLADENOVIC N.(1998). An Introduction to Variable Neighborhood Search. In S. Voss et al. (eds.), Metaheuristics Advances and Trends in Local Search Paradigms for Optimization, pp.433-458, MIC-97, Kluver, Dordrecht.

HANSEN P. and MLADENOVIC N. (2002). An Introduction to Variable Neighborhood Search. En Handbook of Applied Optimization, P.M. Pardalos and M.G.C. Resende (Eds.), pp. 221-234. Oxford University Press.

HOLLORAN T. and BYRN J. (1986). Unitied Airlines Station Manpower Planning System. Interfaces, Vol. 16, nº 1, pp.39-50.

HOLLAND J. (1975). Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Abor.

KEITH E.G. (1979). Operator Scheduling . AIIE Transactions, 11, 1, pp. 37-41.

KIRKPATRICK S., GELATT JR C. D. and VECCHI M.P. (1983). Optimization by Simulated Annealing Science. Vol. 220, 1983. Pp. 671-680.

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

MLADENOVIC N., y HANSEN, P. (1997). Variable Neighborhood Search. Computer and Operations Research, vol.24,n 11, pp.1097-1100.

MOONDRA S.L. (1976). An L.P. Model for Work Force Scheduling for Banks. J. Bank Res., 7, 4, pp.299-301.

MOSCATO P. (1989). On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Toward Memetic Algorithms. Caltech Concurrent Computation Program, C3P Report 826.

MOSCATO P. (2002). Variable Neighborhood Search, in Handbook of Applied Optimization, P. M. Pardalos and M. G. C. Resende (Eds.), Oxford University Press, pp. 221-234.

MOSCATO P. and COTTA C. (2002). A Gentle Introduction to Memetic Algorithms, to appear in Handbook of Metaheuristics. Kluwer.

PITSOULIS L. S. and RESENDE M.G.C. (2002). Greedy Randomized Adaptive Search Procedures. En Handbook of Applied Optimization, P.M. Pardalos and M.G.C. Resende (Eds.), pp. 168-182. Oxford University Press.

QUINN P., ANDREWS B. and PARSONS H. (1991). Allocating Telecommunications Resources at L.L. Bean. Interfaces, Vol. 21, nº 1, pp. 75-91.

TAYLOR P. and HUXLEY S.(1989). A Break from Tradition for the San Francisco Police: Patrol Officer Scheduling Using an Optimization-Based Decision Support System. Interfaces, Vol. 19, nº 1, pp. 4-24.

THOMPSON G.M. (1990). Shift Scheduling when Employees Have Limited Availability: an L.P. Approach. J. of Oper. Management, 9, 3, pp. 352-370.

THOMPSON G.M. (1992). Improving the Utilization of Front-Line Service Delivery System Personnel. Decision Sciences, 23, 5, pp. 1072-1098.

THOMPSON, G.M. (1995). Improved implicit optimal modeling of the labor shift scheduling problem. Management Science , 41 (4), pp. 595-607.

THOMPSON, G.M. (1996). A Simulated Annealing Heuristic for Shift Scheduling Using non-Continuously Available Employees. Computers and Operations Research, Vol. 23, nº 3, pp. 275-278.

THOMPSON, G.M. (1997). Labor Staffing and Scheduling Models for Controlling Service Levels. Naval Research Logistics, vol.44, pp. 719-740.