Curso 2011/12
CIENCIAS Y TECNOLOGÍAS/19
I.S.B.N.: 978-84-15910-14-5

MANUEL VICENTE CENTENO ROMERO

Un estudio metodológico y computacional del problema de asignación multidimensional

Director
JUAN JOSÉ SALAZAR GONZÁLEZ

SERVICIO DE PUBLICACIONES
UNIVERSIDAD DE LA LAGUNA

SOPORTES AUDIOVISUALES E INFORMÁTICOS
Serie Tesis Doctorales

A:

†Delia mi mamá,
mi esposa Milagros,
mis hijos Milgladys Belén y Manuel Vicente,
mis hermanos, hermanas, sobrinas y sobrinos,
la memoria de Vicente y Belén, mis abuelos

Agradecimiento

Deseo expresar mis más sincero agradecimiento a:

Mi familia, en primera instancia, por su comprensión, apoyo y ayuda incondicional durante mis estudios doctorales, sobre todo durante mi estadía en Tenerife-España.

Mi tutor-asesor, Dr. Juan José Salazar González, por su ayuda, sabios comentarios y consejos, comprensión y colaboración en el desarrollo de estas memorias.

La Universidad de Oriente, por parte de la Coordinación de Recursos Humanos, a sus trabajadores(as) y Coordinadores(as) (Prof. Judith de Villarroel y Prof. Russvelt Noriega).

La Universidad de La Laguna, en especial a los Departamentos de Análisis Matemático y Estadística e Investigación de Operaciones, así como a sus profesores y profesoras.

Los doctores Rodrigo Martínez (UDO) y Fernando Pérez González (ULL), Coordinadores, por cada una de sus instituciones, de este convenio.

Todos mis compañeros de estudio y mis amigos venezolanos y españoles residentes en Tenerife.

Índice general

Prólogo

Entre los problemas que estudia la optimización combinatoria están el problema de transporte y el problema de asignación. El problema de asignación (AP) es un caso particular del problema de transporte, donde todas las ofertas y demandas son iguales a uno; y ambos conjuntos, tanto el de oferta como el de demanda, son de igual tamaño.

El AP es considerado un problema polinomial, ya que existen algoritmos exactos ejecutables en tiempo polinomial que lo resuelven hasta la optimalidad. Sin embargo, y debido a la extensa aplicabilidad del AP, se han planteado problemas más complejos, como por ejemplo determinar el coste mínimo de asignar individuos a diferentes trabajos y en determinados horarios. En este trabajo se trata de una extensión del AP conocida como el problema de asignación multi-dimensional (mAP). Este problema aparece en diversas situaciones de la vida real (en ciencia, manufactura, administración, educación, entre otras) y es el objeto de estudio, junto a sus métodos de resolución, en esta memoria.

El *3AP* ha estado presente desde hace bastante tiempo en la literatura (véanse, por ejemplo, Pierskalla, 1968), presentándose en las últimas décadas novedades sobre su estudio y resolución. Entre estos trabajos destacan en forma especial los realizados por Balas y Saltzman (1989 y 1991), Crama y Spieksma (1992), Burkard y Rudolf (1993), Magos y Miliotis (1994), Burkard

et al. (1996a y 1996b), Magos (1996). Más recientemente, han aparecido otros trabajos entre los cuales se encuentran el de Aiex *et al.* (2001) y Huang y Lim (2006).

El mAP es un problema con complejidad algorítmica de tipo NP-difícil, en el sentido fuerte, cuando $m \geq 3$ [Frieze, 1983]. Este hecho, junto con su importancia en operaciones prácticas justifican la necesidad de un estudio profundo que conduzca a la propuesta y desarrollo de nuevos algoritmos para su resolución.

Bandelt *et al.* (1994) habla en su título del mAP, pero el desarrollo del trabajo es para $m = 3$. Sin embargo, otros trabajos tales como: Robertson (2001), Pasiliao (2003), Oliveira y Pardalos (2004), Gutin y Karapetyan (2009a y 2009b) y Karapetyan y Gutin (2010), presentan algoritmos metaheurísticos que resuelven el mAP, para $m \geq 3$.

Existen en la literatura otras investigaciones relacionadas con el mAP, las cuales desarrollan otros tipos de algoritmos, estudios y métodos, tales como: Poore y Robertson (1997) presentan un algoritmo basado en relajación lagrangiana para una clase del mAP, la cual consiste en problemas de observaciones particionadas de examen cuidadoso múltiple de la región de vigilancia y de sensores de huellas y falsas alarmas, sencillos o múltiples. Grundel *et al.* (2004) muestran propiedades asintóticas de problemas de asignación multidimensional aleatorios, a través de la prueba de dos conjeturas. Grundel y Pardalos (2005) presentan un método de generación de un mAP-axial de tamaño controlable con una única solución conocida e investigan ciertas características del mAP generado. Grundel *et al.* (2007) realizan un estudio sobre el número de mínimos locales para el mAP y establecen que el número más grande de mínimos locales tiene estadísticamente un efecto negativo significante en la calidad de soluciones producida por varios algoritmos heurísticos

que involucran la búsqueda de vecindades locales y Krokhmal *et al.* (2007) responden la pregunta de la conducta asintótica del valor óptimo esperado del mAP de gran escala, cuyos costes de asignación son valores independientes aleatorios idénticamente distribuidos con una distribución de probabilidad establecida. Demuestran que para una clase amplia de distribuciones continuas, el valor límite del coste óptimo esperado del mAP es determinado por la localización del punto final izquierdo del conjunto soporte de la distribución y construyen las cotas asintóticas para el coste óptimo esperado.

Dentro del caso tridimensional caben distinguirse dos variantes. En una primera versión se debe elegir cada valor de una dimensión una única vez, y crear así n ternas disjuntas, donde n es el tamaño de la dimensión. Es la llamada versión *axial* (*3AP*-axial). En otra versión se debe elegir un valor de cada dimensión para cada pareja de valores de las otras dos dimensiones. Es la versión *planar* (*3AP*-planar).

En esta memoria hemos desarrollado, con el fin de buscar soluciones al *3AP*-axial y al *3AP*-planar, algunos algoritmos heurísticos y metaheurísticos en búsqueda de "buenas" cotas superiores, y por que no del óptimo, sobre los problemas que se han considerado para su resolución.

En esta investigación también se trató el problema de asignación para $m > 3$, específicamente $m = 4$. Se desarrolla un algoritmo metaheurístico para la resolución del *4AP*, haciendo uso de la metaheurística sistema hormiga.

La memoria se presenta dividida en seis capítulos de la siguiente forma:

Capítulo 1. Primeramente se expone una motivación y luego se define el problema de asignación y varias representaciones de éste. Seguidamente se presenta el *3AP*, los tipos de problemas existentes y sus modelos matemáticos, para luego presentar el *4AP*. Finalmente, se describe el mAP y mencionan

algunos resultados existentes en la literatura.

Capítulo 2. En este capítulo presentamos los diferentes métodos de resolución existentes para problemas de optimización combinatoria de tipo *NP*-difícil. Entre éstos tenemos los algoritmos de relajación, el algoritmo de ramificar y acotar y los algoritmos heurísticos y/o metaheurísticos.

Capítulo 3. Aquí describimos los algoritmos genéticos desarrollados en esta memoria, para la resolución del *3AP*-axial, y presentamos los resultados computacionales sobre problemas generados aleatoriamente y problemas existentes en la literatura. Los resultados principales han sido publicados en González y Centeno (2001).

Capítulo 4. Este capítulo se desarrolla de igual forma que el anterior, con la salvedad que el algoritmo que se presenta hace uso de la metaheurística búsqueda tabú. Los resultados han sido publicados en Centeno (2007) y Centeno et al. (2010).

Capítulo 5. En este capítulo se desarrolla un algoritmo heurístico para la resolución del *3AP*-planar y presentamos los resultados computacionales sobre los problemas tratados.

Capítulo 6. En este capítulo se desarrolla un algoritmo para resolver el *4AP* haciendo uso de la metaheurística sistema hormiga y se muestran los resultados computacionales sobre los problemas tratados. Los resultados han sido publicados en Centeno y Salazar (2008).

Finalmente, se muestran las conclusiones y algunas recomendaciones, así como las referencias bibliográficas.

Este trabajo se inicia en el año de 1998, cuando el Departamento de Análisis Matemático de la Universidad de La Laguna (ULL) realiza un convenio con varias universidades venezolanas, entre ellas la Universidad de Oriente (UDO), en el cual docentes en el área de Matemáticas de esta última institu-

ción realizarían sus estudios doctorales en este departamento. Es por ello que un grupo de docentes de la ULL se trasladaron entre los años 1998 y 1999 a la UDO para dictar las asignaturas correspondientes al Doctorado en Análisis Matemático de la ULL. Luego los doctorandos estuvimos trabajando entre cuatro y cinco meses de cada año en la ULL y el resto del año, con una carga académica mínima, en nuestros respectivos lugares de trabajo en Venezuela.

Durante estos años de mis estudios doctorales, hemos realizado las siguientes publicaciones:

1. González S., J. y Centeno R., M. V. (2001). Desarrollo de un programa para resolver el problema de asignación 3-dimensional, a través de un algoritmo genético. *SABER* 13(2):123-126.

2. Marval, F., Centeno R., M. V. y Salazar G., J. J. (2004). Heurístico para balancear una línea de ensamblaje simple (SALBP, Problema tipo-1). *Div. Matemáticas* 12(1):25-34.

3. Centeno R., M. V. (2007). Búsqueda de soluciones para el 3AP-axial, usando búsqueda por entornos. *FARAUTE* 2(1):20-27.

4. Centeno R., M. V. y Salazar S., H. A. (2008). Algoritmo para el 4AP haciendo uso de la Metaheurística Sistema Hormiga. *Rev. Ing. Industrial* 7 (2):73-83.

5. Centeno R., M. V., Urbina C, J., Salazar G., J. J. (2010). Un algoritmo de búsqueda tabú para resolver el problema de asignación 3-dimensional axial. *ICHIO* 1(1):35-45.

También he asistido, en calidad de ponente mostrando algunos resultados relacionados con esta investigación, a varios congresos nacionales e internacionales, tales como:

1. X Congreso Latinoamericano de Investigación de Operaciones (X CLAIO). 2000. México D.F. México.
2. L Convención Anual ASOVAC. USB. 2000. Caracas-Venezuela.
3. XIV Jornada de Matemáticas. UCLA. 2001. Barquisimeto-Venezuela.
4. IV Congreso Científico de la UDO. UDO. 2001. Cumaná-Venezuela.
5. XV Jornada Venezolana de Matemáticas. LUZ. 2002. Maracaibo-Venezuela.
6. IV Jornada Nacional de Investigación de Operaciones. ULA. 2002. Mérida-Venezuela.
7. XI CLAIO. 2002. Concepción-Chile.
8. IV Workshop ALIO/EURO. 2002. Pucón-Chile.
9. XVIII Congreso de Ecuaciones Diferenciales y Aplicaciones- VIII Congreso de Matemáticas Aplicada (XVIII CEDYA). 2003. Tarragona-España.
10. LIII Convención Anual ASOVAC. LUZ. 2003. Maracaibo-Venezuela.
11. XII CLAIO. 2004. La Habana-Cuba.
12. V Congreso Científico de la UDO. UDO. 2004. Ciudad Bolívar-Venezuela.
13. XVIII Jornada Venezolana de Matemáticas. UCV. 2005. Caracas-Venezuela.
14. I Jornada Tecnológica PDVSA-Sucre. PDVSA-FUNDACITE-UDO-IUT-Cumaná. 2005. Cumaná-Venezuela.
15. V ALIO/ EURO. 2005. París-Francia.
16. XIX Jornada Venezolana de Matemáticas. UDO. 2006. Cumaná-Venezuela.

17. V Jornada de Aplicaciones Matemáticas. UC. 2006. Valencia-Venezuela.

18. VI Congreso Científico de la UDO. UDO. 2006. Puerto La Cruz-Venezuela.

19. XIII CLAIO. 2006. Montevideo-Uruguay.

20. XX Jornada Venezolana de Matemáticas. USB. 2007. Caracas-Venezuela.

21. XXI Jornada Venezolana de Matemáticas. UCLA. 2008. Barquisimeto-Venezuela.

22. XIV CLAIO. 2008. Cartagena-Colombia.

23. XV CLAIO y ALIO-INFORMS Joint International Meeting. 2010. Buenos Aires -Argentina

Cabe destacar la asistencia a los cursos del área de Programación Matemática realizados en la Universidad Miguel Hernández, Elche-España, en el año 2004.

Deseo expresar en esta memoria mi agradecimiento a la UDO, a la ULL y a los profesores Rodrigo Martínez y Fernando Pérez González, Coordinadores de este convenio en representación de la UDO y de la ULL, respectivamente, por la oportunidad brindada y toda la colaboración prestada. De igual forma, quiero agradecer a las siguientes dependencias e instituciones: Coordinación de Recursos Humanos y el Consejo de Investigación de la UDO, FONACIT, FUNDACITE-Sucre por la ayuda económica suministrada para la realización de mis estudios doctorales y la asistencia a los diferentes eventos, tanto nacionales como internacionales, así como a la ULL, quien me dio la oportunidad de asistir a cursos en el área de la Programación Matemática realizados en la Universidad Miguel Hernández en Elche.

Capítulo 1

Introducción: el problema de asignación

1.1. Motivación

Los orígenes de la optimización son muy antiguos, ya que todas las sociedades organizadas se han planteado la necesidad de optimizar el uso de recursos y la planificación de las actividades a realizar.

La optimización pretende maximizar o minimizar una función de varias variables sujeta a restricciones. En innumerables actividades de la economía, la ciencia, la educación, entre otras áreas, se presentan situaciones sujetas a ciertas limitaciones que deben ser analizadas de la mejor manera posible para reducir al mínimo los costes, o bien, para obtener el mayor nivel de ganancia o satisfacción.

Dentro de las Matemáticas Aplicadas existe un campo orientado al diseño de metodologías para resolver, desde el punto de vista práctico y haciendo uso de un ordenador, problemas de optimización con recursos limitados. Este campo es denominado *Programación Matemática*.

La Programación Matemática se divide a su vez en varias ramas según las características de las variables y de las ecuaciones y/o inecuaciones que describen los modelos matemáticos. Entre estas ramas está la *Optimización Combinatoria* que persigue la resolución de problemas de optimización caracterizados por tener un número finito (aunque en general muy grande) de posibles soluciones.

El hecho distintivo de la optimización combinatoria, discreta o entera, es que se exige que algunas de las variables deben pertenecer a un conjunto discreto, típicamente un subconjunto de enteros. Estas restricciones discretas permiten la representación matemática de fenómenos o alternativas donde la indivisibilidad es requerida o donde no existen alternativas de continuidad [Nemhauser y Wolsey, 1998].

Los problemas de optimización combinatoria abundan en la vida cotidiana. Un área de aplicaciones importante y extensa concierne la administración y uso eficiente de escasos recursos para aumentar la productividad. Estas aplicaciones incluyen problemas operacionales como la distribución de bienes, producción *shedulling* y secuenciación de máquinas, problemas de planificación y diseño de redes de telecomunicaciones y transporte, entre otros [Nemhauser y Wolsey, 1998].

Las aplicaciones de optimización combinatoria se han desarrollado rápidamente debido al uso de ordenadores y los datos proporcionados por los sistemas de información. Esto es particularmente relevante en el sector industrial y de la economía, donde la competencia y flexibilidad proporcionada por la tecnología hace indispensable la búsqueda de mejores soluciones sobre conjuntos alternativos más grandes y complejos.

1.2. Conceptos básicos

Los problemas de asignación responden la pregunta ¿cómo asignar n nodos orígenes (trabajadores) a n nodos destinos (máquinas) en la mejor forma posible?. Estos problemas contienen dos componentes: la asignación que tiene tras de sí una estructura combinatoria, y la función objetivo que modela la "mejor forma". Matemáticamente, una *asignación* es una función biyectiva de un conjunto finito en sí mismo, es decir, una permutación. Las asignaciones pueden ser modeladas y visualizadas en diferentes formas: cada permutación ϕ del conjunto $N := \{1, 2, ...n\}$ corresponde en una única forma a la matriz de permutaciones $X_\phi := \{x_{ij}\}$ con $x_{ij} := 1$ para $j := \phi(i)$ y $x_{ij} := 0$ para $j \neq \phi(i)$ [Burkard y Çela, 1998].

Definición 1.2.1 *Un grafo G es un objeto que consiste de dos conjuntos V y E llamados conjunto de vértices o nodos y conjunto de aristas, respectivamente, y se denota $G(V, E)$.*

Definición 1.2.2 *Sea G un grafo. Un camino en G es una sucesión de nodos y aristas $v_0, a_1, v_1, a_2, ..., a_n, v_n$, tal que los extremos de la arista a_i son v_{i-1}, v_i, $i := 1, ..., n$, $n \geq 0$.*

Definición 1.2.3 *Un camino se dice cerrado si los nodos v_0 y v_n coinciden.*

Definición 1.2.4 *Un camino cerrado de un grafo G que pasa por cada nodo de G, excepto por el inicial, una y sólo una vez se denomina un camino hamiltoniano.*

Definición 1.2.5 *Un tour es un camino hamiltoniano.*

Definición 1.2.6 *Un grafo G se llama hamiltoniano si contiene un tour.*

Definición 1.2.7 *Dos nodos u y v de un grafo G se dicen que están conectados cuando existe un camino en G de extremos u y v.*

Definición 1.2.8 *Sea G un grafo. Si para todo par de nodos u y v de G existe un camino en G que los conecta, entonces se dice que el grafo G es conexo.*

Definición 1.2.9 *Un grafo conexo que no tiene ciclos se denomina árbol.*

Definición 1.2.10 *Un árbol que conecta todos los vértices de un grafo G se denomina un árbol de expansión de G.*

Definición 1.2.11 *Dos nodos de un grafo G son adyacentes si existe una arista de G que los contiene.*

Definición 1.2.12 *El grafo G se dice completo si todos sus nodos son adyacentes dos a dos.*

Definición 1.2.13 *Si $G(V, E)$ es un grafo y $V := \{1, 2, ..., n\}$, se llama matriz de adyacencia del grafo G y se denota $M_a(G)$, a la matriz $M_a(G) := [a_{ij}; i, j \in \{1, 2, ..., n\}]$, donde a_{ij} es igual al número de aristas cuyos extremos son i y j.*

Definición 1.2.14 *Un grafo $G'(V', E')$ se dice que es un subgrafo del grafo $G(V, E)$ si $V' \subseteq V$ y $E' \subseteq E$.*

Definición 1.2.15 *Un "clique.^{en} el grafo G es un subgrafo completo maximal, es decir, un subgrafo $G'(V', E')$ con*

$$E\text{':=}\{(i, j) : i \in V' y j \in V'\}$$

Definición 1.2.16 *Un grafo es k-partito si sus nodos pueden ser particionados en k subconjuntos tales que dos nodos en el mismo conjunto no están unidos por una arista.*

Definición 1.2.17 *Un grafo k-partito se dice completo si todo nodo es adyacente a todos los otros nodos excepto a aquéllos en su subconjunto.*

El grafo k-partito completo con n_i nodos en su i-ésimo subconjunto, es denotado por $K_{n_1,n_2,\ldots,n_i,\ldots,n_k}$. La matriz de permutaciones X_ϕ que representa una asignación puede visualizarse como la matriz de adyacencia de un grafo bipartito (2-partito) $G_\phi(V \cup W, E)$, donde los conjuntos de nodos V y W tienen n elementos, es decir, $|V| := n := |W|$, y existe una arista $(i, j) \in E$ con $j := \phi(i)$, como se representa en la Figura 1.1.

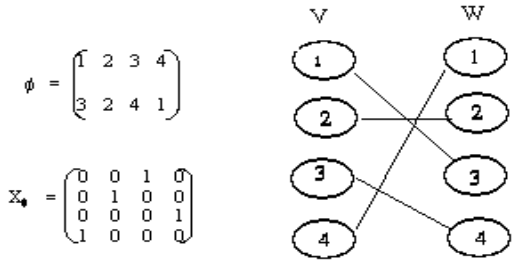

Figura 1.1: Diferentes representaciones de asignaciones.

Definición 1.2.18 *Un grafo bipartito completo de la forma $K_{1,s}$ se denomina un grafo estrella.*

Como todo grafo estrella G se puede representar como un árbol que conecta todos los nodos de G, entonces G es un árbol de expansión al que se denomina estrella de expansión.

El conjunto de todas las asignaciones de n artículos (orígenes y destinos) es denotado por S_n y tiene $n!$ elementos. Se puede describir este conjunto por las siguientes ecuaciones llamadas *restricciones de asignación.*

$$\begin{aligned} &\sum_{i=1}^{n} x_{ij} := 1, \quad j := 1, \ldots, n \\ &\sum_{j=1}^{n} x_{ij} := 1, \quad k := 1, \ldots, n \\ &x_{ij} \in \{0, 1\}\ , \quad i := 1, \ldots, n; \quad j := 1, \ldots, n; \end{aligned} \tag{1.1}$$

Este problema tiene gran cantidad de aplicaciones prácticas y una gran importancia teórica. Además, es de sencilla formulación y existen algoritmos eficientes para su resolución exacta (variantes polinómicas del método simplex como el clásico algoritmo húngaro) [Burkard y Çela, 1998].

Definición 1.2.19 *Supónganse conocidos los c_{ij}. Si cada origen sólo se puede asignar a un destino y a cada destino se le puede asignar un único origen, entonces el problema de determinar qué origen debe asignarse a cada destino, de manera que el coste total resulte mínimo, se denomina Problema de Asignación (AP).*

Un modelo matemático para este problema en el cual están representadas las dos componentes de asignación (la función objetivo y las ecuaciones (1.1)) está dado por el modelo (1.2).

Obviamente, las soluciones de un AP se pueden representar gráficamente mediante arcos disjuntos de un grafo bipartito de $2n$ nodos, como se muestra en la Figura 1.2.

(1.2)

$$\text{mín} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

sujeto a:

$$\sum_{j=1}^{n} x_{ij} := 1, \qquad i := 1, \ldots, n$$

$$\sum_{i=1}^{n} x_{ij} := 1, \qquad j := 1, \ldots, n$$

$$x_{ij} \in \{0, 1\}\ , \qquad i := 1, \ldots, n; \quad j := 1, \ldots, n$$

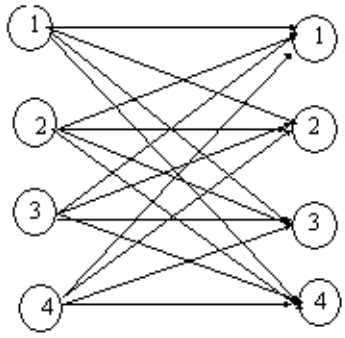

Figura 1.2: Grafo bipartito con n:=4 (8 nodos).

Definición 1.2.20 *Una matriz real C de orden $m \times n$ se llama una matriz Monge si:*

$$c_{ij} + c_{rs} \leq c_{is} + c_{rj}; \quad \textit{para todo } 1 \leq i < r \leq m, 1 \leq j < s \leq m \qquad (1.3)$$

Definición 1.2.21 *Una matriz real C de orden $m \times n$ se llama una matriz Monge inversa, si C satisface la inversa de la propiedad (1.3), es decir:*

$$c_{ij} + c_{rs} \geq c_{is} + c_{rj}; \quad \textit{para todo } 1 \leq i < r \leq m, 1 \leq j < s \leq m \qquad (1.4)$$

Es importante notar que la propiedad (1.3) tiene una larga historia. Ya en 1781 el ingeniero y matemático francés Gaspard Monge (1746-1818) la consideró en el problema relacionado con el transporte de tierra [Burkard et al., 1996a].

Monge quiso partir dos volúmenes igualmente grandes (representando la situación inicial y final de la Tierra transportada) en infinitamente muchas partículas pequeñas y asociarlas entre sí, para que la suma de los productos de las longitudes de los caminos usados, multiplicados por el volumen de las partículas, se minimice. Este problema puede verse como el problema de transporte continuo especial [Burkard et al., 1996a].

En el problema de transporte *(TP)* es indiferente si la matriz de coste es una matriz de Monge o Monge inversa. Ello se debe a que todos los resultados donde C es una matriz de Monge pueden trasladarse a matriz Monge inversa intercambiando el orden de las columnas de la matriz [Burkard et al., 1996a].

Luego, en el AP, por ser un caso especial del TP, sucede de igual forma. Además, si la matriz de costes C es una matriz Monge, entonces tiene una solución óptima con una estructura fija que no depende de C.

1.3. El problema de asignación 3-dimensional

Si se consideran ahora tres conjuntos de nodos I, J, K, tales que $|I| := |J| := |K| := n$, $I \cap J := J \cap K := I \cap K := \phi$, y dada la familia

$$S := \{(i, j, k) : i \in I, j \in J, k \in K\},$$

se desea hallar la partición de S que cubra exactamente la unión de los tres conjuntos, y además el coste de asignación sea mínimo. Este problema es denominado *Problema de Asignación 3-dimensional axial* ($3AP$-axial), para distinguirlo de otro problema de asignación 3-dimensional conocido como $3AP$-planar, y que se define a continuación.

Considérense n^3 coeficientes de costes c_{ij} dados, los cuales denotan el coste que origina asignar el nodo $i \in I$ al nodo $j \in J$ y éste a su vez al nodo $k \in K$. El $3AP$-planar consiste en encontrar el costo mínimo de la colección de un conjunto de n^2 tripletas que no tengan en común ninguna pareja de elementos, es decir, ningún par de tripletas de la solución pueden tener dos componentes iguales. El $3AP$-planar está dado por el siguiente modelo matemático.

$$\begin{aligned}
&\text{mín} \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} c_{ijk}x_{ijk} \\
&\text{sujeto a:} \qquad (1.5)\\
&\sum_{i=1}^{n} x_{ijk} := 1, \quad j := 1,\ldots,n; \quad k := 1,\ldots,n \\
&\sum_{j=1}^{n} x_{ijk} := 1, \quad i := 1,\ldots,n; \quad k := 1,\ldots,n \\
&\sum_{k=1}^{n} x_{ijk} := 1, \quad i := 1,...,n, \quad j := 1,...,n \\
&x_{ijk} \in \{0,1\}, \quad i := 1,\ldots,n; \quad j := 1,\ldots,n; \quad k := 1,\ldots,n.
\end{aligned}$$

Una representación geométrica del modelo (1.5) se presenta en la Figura 1.3.

Definición 1.3.1 *Un rectángulo latino es una matriz de orden $m \times n$ con elementos $a_{ij} \in \{1,...,n\}$ tal que los enteros en cada fila y columna sean distintos.*

Definición 1.3.2 *Un cuadrado latino de orden n es un rectángulo latino con $m := n$. Específicamente, un cuadrado latino consiste de n conjuntos de los números desde 1 hasta n arreglados de tal forma que ninguna fila o columna contiene el mismo número dos veces.*

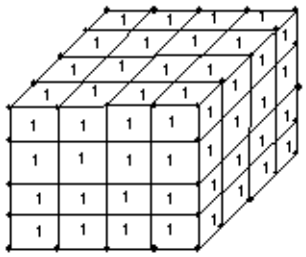

Figura 1.3: Una representación geométrica del *3AP*-planar para $n = 4$.

El número $N(n, n)$ de cuadrados latinos de orden $n := 1, 2, 3, 4, 5, ...$ es 1, 2, 12, 576, 161280,... [Sloane, 2003].

Cada piso en el arreglo 3-dimensional, mostrado en la Figura 1.4, x_{ijk}, debe contener una asignación (2-dimensional). Así, las soluciones factibles del $3AP$-planar corresponden a un cuadrado latino. La Figura 1.4 muestra una solución factible para el $3AP$-planar con $n := 4$, donde el número 1 representa la asignación en el piso horizontal inferior, el número 2 muestra la asignación en el segundo piso, el número 3 representa la asignación en el tercer piso y el número 4 la asignación en el piso superior. Debido a esta interpretación, el número de soluciones factibles del $3AP$-planar de tamaño n, es igual al número de cuadrados latino de orden n, y por lo tanto crece muy rápido.

Frieze (1983), demostró que el $3AP$-planar es NP-difícil. No se conocen muchos algoritmos para el $3AP$-planar. El primer algoritmo de ramificar y acotar fue realizado por Vlach (1967). Vlach computó las cotas inferiores por aplicaciones (generalizadas) de reducciones de filas y columnas. Otro algoritmo de ramificar y acotar para el *3AP*-planar, fue descrito por Magos

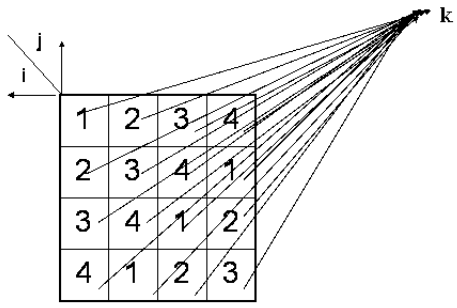

Figura 1.4: Un cuadrado latino representando una solución factible del *3AP* planar, con n = 4.

y Miliotis (1994).

Consideremos nuevamente n^3 coeficientes de costes c_{ijk} y describamos ahora el modelo correspondiente al $3AP$-axial.

$$
\begin{aligned}
&\text{mín} \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} c_{ijk}x_{ijk} \\
&\text{sujeto a:} \\
&\sum_{j=1}^{n}\sum_{k=1}^{n} x_{ijk} := 1, \quad i := 1,\ldots,n \\
&\sum_{i=1}^{n}\sum_{k=1}^{n} x_{ijk} := 1, \quad j := 1,\ldots,n \\
&\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijk} := 1, \quad k := 1,\ldots,n \\
&x_{ijk} \in \{0,1\}\ , \quad i := 1,\ldots,n; \quad j := 1,\ldots,n; \quad k := 1,...,n.
\end{aligned}
\tag{1.6}
$$

El origen del nombre del modelo se debe al asignar los 1s del lado derecho de las restricciones en el modelo (1.6) a las distintas posiciones en los ejes de un arreglo 3-dimensional. La suma sobre el "piso" correspondiente en el arreglo tiene que ser igual a la asignación a la posición de los ejes, como se

muestra en la Figura 1.5.

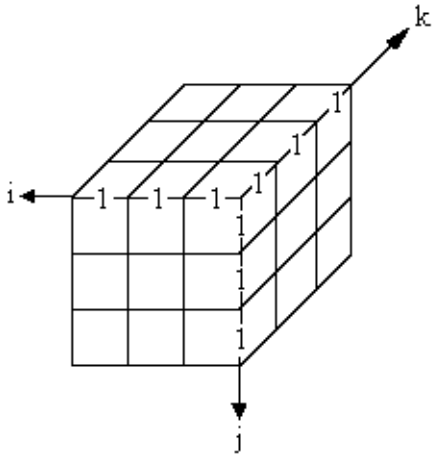

Figura 1.5: Una representación geométrica del *3AP*-axial para $n = 4$.

.

Ahora, $x_{ijk} := 1$ si el trabajo j es asignado al trabajador i en la máquina k, y $x_{ijk} := 0$ en otro caso. Una solución factible de este problema será un arreglo 0-1 que satisface las restricciones de asignación dadas en (1.4).

Equivalentemente, el $3AP$-axial puede escribirse con la ayuda de dos permutaciones ϕ y ψ

$$\min_{\phi,\psi \in S_n} \sum_{i=1}^{n} c_{i\phi(i)\psi(i)} \tag{1.7}$$

Así, este problema tiene $(n!)^2$ soluciones factibles. Karp (1972) demostró que el $3AP$-axial es NP-difícil.

En términos de Teoría de Grafos, el $3AP$-axial puede definirse como sigue: dados un grafo tripartito completo $K_{n,n,n}$ con conjuntos de vértices disjuntos U, V y W, hallar un subconjunto T de n triángulos, $T \subseteq U\text{x}V\text{x}W$, tal que todo elemento de $U \cup V \cup W$ ocurre en exactamente un triángulo de T y el costo total

$$\sum_{(i,j,k)\in T} c_{ijk}$$

es minimizado.

Considérese el problema de programación lineal dado a continuación.

$$\begin{aligned} &\text{mín } z := \mathbf{cx} \\ &s.a : A\mathbf{x} \leq \mathbf{b}, \end{aligned} \tag{1.8}$$

donde A es una matriz de orden mxn y $\mathbf{b}$ un m-vector.

Definición 1.3.3 *Si $\boldsymbol{c}$ es un vector tal que $\boldsymbol{c} \neq 0$ y $\delta := \text{máx}\{\boldsymbol{cx} : A\boldsymbol{x} \leq \boldsymbol{b}\}$, el hiperplano afín $\{\boldsymbol{x} : \boldsymbol{cx} := \delta\}$ es llamado un hiperplano soporte del poliedro $P := \{\boldsymbol{x} : A\boldsymbol{x} \leq \boldsymbol{b}\}$.*

Definición 1.3.4 *Un subconjunto F de P es llamado una cara de P si $F := P$ o si F es la intersección de P con un hiperplano soporte de P.*

Definición 1.3.5 *Una faceta de P es una cara maximal distinta de P.*

Si $F := \{\mathbf{x} \in P : a_i\mathbf{x} := \beta_i\}$, se dice que $a_i\mathbf{x} := \beta_i$ define o determina la faceta F, donde $a_i\mathbf{x} := \beta_i$ es un hiperplano generador de P.

Lema 1.3.6 *Cada cara de P, excepto P misma, es la intersección de facetas de P.*

Demostración. Por las definiciones 1.3.5 y 1.3.4, cada hiperplano generador de P define una faceta, así la intersección de facetas está dada por:

$$F := \{\mathbf{x} \in P : A'\mathbf{x} := \mathbf{b'}\},$$

para algún subsistema A'$\mathbf{x} \leq$ **b'** de A$\mathbf{x} \leq \mathbf{b}$, y por 1.3.4, una cara G es la intersección de P con un hiperplano soporte H de P, donde **c** es la suma de las filas de *A'*, así si $\mathbf{x} \in F$, entonces $\mathbf{x} \in P$ y $\mathbf{x} \in H$, ahora si $\mathbf{x} \in P \cap H$, entonces $\mathbf{x} \in F$, luego $F := P \cap H$, de donde F es una cara de P. ■

Definición 1.3.7 *Un "cliqueçubierto exactamente, es un conjunto de "cliques", $(i, j, k) \in IxJxK$, que particionan el conjunto de nodos y representan una cara del 3AP.*

En términos de $K_{n,n,n}$, en el *3AP* la matriz A es la matriz de incidencia de nodos versus cliques, la cual tiene una fila por cada nodo y una columna por cada clique de $K_{n,n,n}$.

Definición 1.3.8 *El grafo intersección $G_A(V, E)$ de la matriz A, es el grafo clique intersección de $K_{n,n,n}$, el cual tiene un nodo para cada clique de $K_{n,n,n}$ y una arista para cada par de cliques que comparten algún nodo de $K_{n,n,n}$. Representando así una faceta del 3AP.*

Definición 1.3.9 *Se llama cápsula convexa de soluciones factibles del 3AP, a:*

$$P_I := conv\left\{\boldsymbol{x} \in \{0,1\}^{n^3} : A\boldsymbol{x} \leq \boldsymbol{1}\right\}.$$

Definición 1.3.10 *Un politopo está definido como la intersección finita de subespacios cerrados.*

Definición 1.3.11 *El politopo inducido por el 3AP se define por:*

$$\tilde{P}_I := conv\left\{\boldsymbol{x} \in \{0,1\}^{n^3} : A\boldsymbol{x} \leq \boldsymbol{1}\right\}.$$

En 1987 Euler inició la investigación del politopo de asignación 3-índice axial (esto es la cápsula convexa de soluciones factibles del modelo (1.6) considerando el papel de agregar ciclos de una clase de facetas de este politopo). Independientemente, Balas y Saltzman (1989) investigan en detalle la estructura poliédrica del politopo de asignación 3-índice, y mostraron que este politopo tiene dimensión $n^3 - 3n + 2$. Ellos describieron un algoritmo de separación de $O(n^4)$ para facetas inducidas por ciertos cliques.

Balas y Qi (1993) continuaron este trabajo y presentaron algoritmos de separación de $O(n^3)$. Identificaron facetas de primera clase tan buenas como para una nueva clase de facetas (nótese que como existen n^3 variables, estos son algoritmos en tiempo lineal en el número de variables).

Otro trabajo relacionado con la búsqueda de algoritmos para el *3AP*, es el realizado por Aiex *et al.* (2001), en el cual describen variantes del GRASP con encadenamiento de trayectorias para el *3AP*, mejorando las soluciones de las heurísticas propuestas por Balas y Saltzman (1991), Crama y Spieksma (1992) y Burkard *et al.* (1996), sobre la gran mayoría de los instances propuestos en estos artículos. De igual forma, Huang y Lim (2006) desarrollan un algoritmo genético híbrido para el problema de asignación 3-índices, ellos proponen una nueva búsqueda local heurística que resuelve el problema simplificándolo al problema de asignación clásico y desarrollan un algoritmo híbrido entre su heurística y un algoritmo genético. Los resultados experimentales de este trabajo indican que el método híbrido es superior a métodos heurísticos anteriores que incluyen aquéllos propuestos por Balas y Saltzman (1991), Crama y Spieksma (1992), Burkard *et al.* (1996) y Aiex *et al.* (2001).

1.4. El problema de asignación 4-dimensional

Supóngase que se tienen n individuos, n trabajos, n máquinas y n insumos, entonces c_{ijkl} es el coste que se incurre al asignar al individuo i en el trabajo j en la máquina k con el insumo l. Se desea encontrar el coste mínimo de asignar los individuos en los trabajos con máquinas establecidas, utilizando diferentes insumos. Sea x_{ijkl} la variable de decisión para este problema, entonces cada solución básica factible $x_{ijkl} := 1$ significa que el i-ésimo individuo en el j-ésimo trabajo con la k-ésima máquina utilizando el l-ésimo insumo es asignado, y $x_{ijkl} := 0$ indica que el i-ésimo individuo en el j-ésimo trabajo con la k-ésima máquina utilizando el l-ésimo insumo no es asignado. Al igual que en el caso 3-dimensional, aqui se presentan dos tipos de problemas, el primero es el caso axial, en el cual una vez que los i-ésimo, j-ésimo, k-ésimo, l-ésimo elementos son asignados, no pueden ser utilizados en otras asignaciones y el caso planar que consiste en encontrar el costo mínimo de la colección de un conjunto de n^2 4-uplas que no tengan en común ninguna pareja de elementos, es decir, ningún par de 4-uplas de la solución pueden tener dos componentes iguales.

Un modelo matemático para este problema es el siguiente:

$$\text{mín} \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{l=1}^{n} c_{ijkl} x_{ijkl}$$

sujeto a: (1.9)

$$
\begin{aligned}
\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} x_{ijkl} &:=1, \quad i := 1,\ldots,n \\
\sum_{i=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} x_{ijkl} &:=1, \quad j := 1,\ldots,n \\
\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{l=1}^{n} x_{ijkl} &:=1, \quad k := 1,\ldots,n \\
\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} x_{ijkl} &:=1, \quad l := 1,\ldots,n \\
x_{ijkl} \in \{0,1\}\ , &\quad i := 1,\ldots,n;\ j := 1,\ldots,n; \\
&\quad k := 1,\ldots,n;\ l := 1,\ldots,n.
\end{aligned}
$$

Este modelo (1.9) pertenece al problema de asignación 4-dimensional, y consiste en: dados 4 conjuntos disjuntos I, J, K, L con cardinalidad n cada uno y un peso asociado a cada elemento ordenado por la 4-upla $(i,j,k,l) \in I \times J \times K \times L$, se desea encontrar una colección de mínimo peso de n conjuntos disjuntos. Sea A la matriz de coeficientes del conjunto de restricciones del $4AP$, la cual es unimodular, ya que todos sus coeficientes son enteros y el determinante de toda submatriz invertible es 0, 1 ó -1, con a_{ijkl} el coeficiente de la matriz A que representa la columna asociada. Luego, a_{ijkl} se representa por: $a_{ijkl} := e_i + e_{n+j} + e_{2n+k} + e_{3n+l}$, para $(i,j,k,l) \in I \times J \times K \times L$ de orden n. Entonces, el conjunto de índices fila y columna de A se da por $|I| + |J| + |K| + |L| := 4n$ y $|I| \times |J| \times |K| \times |L| := n^4$, respectivamente.

Un aumento considerable del valor n, implicaría un incremento en el tamaño de la matriz A, trayendo como consecuencia la utilización de mayor espacio en memoria, y además, mayor tiempo en la realización de las operaciones correspondientes.

Es por lo antes expuesto, que se han diseñado métodos de resolución para este tipo de problemas, tales como las metaheurísticas, los cuales aunque no obtengan la solución óptima, producen una "buena" aproximación a ésta.

1.5. El problema de asignación multi-dimensional

El problema de asignación multi-dimensional *(mAP)*, con $m \geq 3$, es una extensión natural del *AP*. Éste fue considerado por primera vez por Pierskalla (1968).

Cuando el número de conjuntos disjuntos a considerar en el problema de asignación es $m \geq 3$, estamos en presencia de un problema de asignación multidimensional (mAP). Este problema aparece en diversas situaciones de la vida y es descrito en esta sección, siendo un problema con complejidad algorítmica de tipo *NP*-difícil, en el sentido fuerte [Frieze, 1983].

Este hecho, junto con su importancia en operaciones prácticas justifica la necesidad de un estudio profundo que conduzca a la propuesta y desarrollo de nuevos algoritmos para su resolución.

Entre este tipo de problemas se encuentran el *3AP*-axial, el *3AP*-planar y el *4AP*, descritos anteriormente.

La formulación general del *mAP*, está dada por el siguiente modelo.

$$\text{mín} \sum_{i_1=1}^{n} \sum_{i_2=1}^{n} \ldots \sum_{i_m=1}^{n} c_{i_1 i_2 \ldots i_m} x_{i_1 i_2 \ldots i_m}$$

sujeto a:

(1.10)

$$
\begin{aligned}
&\sum_{i_2=1}^{n}\sum_{i_3=1}^{n}\dots\sum_{i_i=1}^{n}\sum_{i_{i+1}=1}^{n}\dots\sum_{i_m=1}^{n} x_{i_1 i_2 \dots i_m} := 1, && i_1 := 1,\dots,n\\
&\sum_{i_1=1}^{n}\sum_{i_3=1}^{n}\dots\sum_{i_i=1}^{n}\sum_{i_{i+1}=1}^{n}\dots\sum_{i_m=1}^{n} x_{i_1 i_2 \dots i_m} := 1, && i_2 := 1,\dots,n\\
&\quad\vdots \quad\quad \vdots \quad\quad \vdots \quad\quad \vdots \quad\quad\quad \vdots && \quad\vdots\\
&\sum_{i_1=1}^{n}\dots\sum_{i_{k-1}=1}^{n}\sum_{i_{k+1}=1}^{n}\dots\sum_{i_m=1}^{n} x_{i_1 i_2 \dots i_m} := 1, && i_k := 1,\dots,n\\
&\quad\vdots \quad\quad \vdots \quad\quad \vdots \quad\quad \vdots \quad\quad\quad \vdots && \quad\vdots\\
&\sum_{i_1=1}^{n}\dots\sum_{i_k=1}^{n}\sum_{i_{i+1}=1}^{n}\dots\sum_{i_{m-1}=1}^{n} x_{i_1 i_2 \dots i_m} := 1, && i_m := 1,\dots,n\\
&\qquad\qquad x_{i_1 i_2 \dots i_m} \in \{0,1\}\ , \quad 1 \le i_1, i_2, \dots, i_m \le n
\end{aligned}
$$

También se puede modelar y visualizar un mAP a través de $m-1$ permutaciones $\phi_1, ...\phi_{m-1}$, donde la función objetivo está dada por:

$$\min_{\phi_1 \dots \phi_{m-1}} \sum_{i_1=1}^{n} c_{i,\phi_1(i)\dots\phi_{m-1}(i)} x_{i,\phi_1(i)\dots\phi_{m-1}(i)}$$

El mAP es un problema de optimización combinatoria, ya que pretende encontrar una solución que minimice una función de costes, siendo finito el número de soluciones posibles. En efecto, hay $(n!)^{m-1}$ posibles asignaciones. Precisamente, lo grande de este número impide resolver la enumeración exhaustiva, inclusive cuando n y m son moderadamente pequeños.

Ahora, en el mAP, el cual es en general un problema NP-difícil, si el arreglo de coeficientes de costes es un arreglo Monge, éste es resuelto por la permutación identidad $\phi_i := id$; para $i := 1, ..., m-1$. Burkard, Rudolf y Woeginger (1996) mostraron, sin embargo, que el mAP, para $m \ge 3$, es NP-difícil si el arreglo de costes es Monge inverso (véase Definición 1.2.21). Esto implica que maximizando la función objetivo con elementos de costes sacados de un arreglo Monge, el mAP es NP-difícil [Burkard et al., 1996b].

Usando la terminología de grafo, el mAP puede ser descrito como sigue:

sea $G(V_1, ..., V_m; E)$ un grafo m-partito completo, con conjunto de vértices V_i, $|V_i| := n$; para $i := 1, ..., m$ y E un conjunto de aristas dado. Un subconjunto X de $V := \cup_{i=1}^{m} V_i$ es un "*clique*", si éste satisface cada conjunto V_i en exactamente un vértice. Una asignación m-dimensional es una partición de V en n "*cliques*"disjuntos dos a dos. Si c es un valor real de la función de coste definida sobre el conjunto de "*cliques*"del grafo $G(V_1, ..., V_m; E)$, se llama coste de una asignación a la suma de los costes de sus n "*cliques*". El problema de asignación m-dimensional pregunta por una asignación m-dimensional con coste mínimo.

Bandelt, Crama y Spieksma (1994) consideraron casos donde el coste c de un "*clique*"no es arbitrario, sino dados como una función de costes elementales asociados a las aristas de un grafo m-partito completo. Los costes de un "*clique*"que ellos consideraron son: la suma de los costes (suma de las longitudes de las aristas inducidas por el "*clique*"), costes estrella (longitud mínima de un estrella en expansión), costes del tour (costo mínimo de un tour del *TSP* en el "*clique*"), y costes del árbol (costo mínimo de un árbol de expansión en el "*clique*"). Los autores obtienen costes de casos peores sobre la razón entre los costes de las soluciones obtenidas por aplicación de heurísticas.

Gutin y Karapetyan (2009a) agrupan a todas las familias de instances en dos clases: instances con pesos independientes e instances con pesos descomponibles. Después muestran que las heurísticas se comportan diferentemente en los casos de estas clases y, así, esta división ayuda en el análisis experimental correcto de los algoritmos de búsquedas locales. Una de las clases más estudiadas de instantes para el *mAP* es la Familia de Instances Aleatorias. En la aleatoriedad, el peso asignado a un vector tiene una distribución aleatoriamente uniforme en el intervalo [a, b-1]. Los casos aleatorios

son usados en [Pierskalla, 1968] [Balas y Saltzman, 1991] [Aiex et al., 2001] [Huang y Lim, 2006] y algunos otros.

Se puede ver, que un instance aleatorio para el $4AP$ tiene una (an)-asignación (asignación de n vectores de peso a) con la probabilidad muy cerca de 1 si $n \geq 40$; un instance para el $5AP$ tiene una (an)-asignación con la probabilidad muy cerca de 1 para $n \geq 12$, un instance para el $6AP$ tiene una (an)-asignación con la probabilidad muy cerca de 1 para $n \geq 8$ y un instance para el $7AP$ tiene una (an)-asignación con la probabilidad muy cerca de 1 para $n \geq 7$; así que, es muy probable que las soluciones óptimas para todos los casos aleatorios usados en los experimentos sean de peso an [Gutin y Karapetyan, 2009a].

En ese artículo, Gutin y Karapetyan (2009a) proponen un nuevo caso especial de peso descomponible, RaizCuadrada. Es una modificación de la familia de instances de clique. Ellos asumen tener s radares y n aviones y cada radar observa todos los aviones. El problema es para asignar señales las cuales vienen de diferentes lugares. Realmente es natural definir una función de distancia entre cada par de señales de diferentes radares, y para un conjunto de señales que corresponden a un avión, la suma de las distancias debe ser pequeña, así veintiseis (26) es una buena opción. Sin embargo, no es realmente correcto minimizar la distancia total entre las señales asignadas, pero también se debe asegurar que ninguna de estas distancias es larga. Algunos requisitos aparecen en varias aplicaciones. Ellos proponen una función de peso, la cual puede llevar a la menor distancia total entre las señales asignadas y la dispersión de las distancias.

Gutin y Karapetyan (2009b) tratan familias de instance usadas para la evaluación experimental del algoritmo memético desarrollado por ellos. Usan pseudos instances reales-mundiales, tales como: la Composición de Ciclos

(CC), Clique Compuesto (CQ), Raíces Cuadradas (SR) y la versión perturbada de cada uno de ellos. La CC y familias de CQ se propusieron antes en la literatura (vea Burkard y Çela (1998) y referencias allí) mientras los otros son tratados en ese artículo por primera vez. Para explicar estas familias de instances es bueno representar el mAP como un problema de grafos. Sea $G = (V_1 \cup V_2 \cup \ldots V_m,\ E)$ un grafo m-partito, donde $|Vi| = n$ para cada i=1,2,...,m. Un peso se asigna a cada clique en G, es decir, a cada subgrafo inducido $(\{v_1, v_2, ..., v_m\}, \{v_1v_2, v_1v_3, ..., v_1v_m, v_2v_3, v_2v_4, ..., v_2v_m, ..., v_{s-1}v_m\})$, donde $v_1 \in V_1$, ..., $v_m \in V_m$; un clique corresponde a un vector en la interpretación original. El objetivo es encontrar n cliques disjuntos, tal que su peso total sea mínimo.

La idea de todo el CC, CQ y SR es que en la vida real es posible usarlo sólo para definir alguna relación entre dos objetos, pero no entre m objetos. Así, el grafo G muestra su peso y el peso de un clique $C = (V_C, E_C)$ debe ser una función de los pesos E_C de las aristas en el clique.

En este caso, el objetivo no es sólo minimizar los pesos considerados, sino también no guardar todos los pesos demasiado grande.

Los casos perturbador son los mismos casos, pero el peso asignado a cada clique se modifica aleatoriamente: $w(c) = w(c) + r$, para cada clique c donde $r \in \{0,1,...,19\}$ es escogido aleatoriamente.

En Gutin y Karapetyan (2009b), se incluyen instance para el *3AP*, *4AP*, *5AP* y el *6AP*, seis (6) tipos de instance para cada valor de m

Capítulo 2

Métodos de resolución

2.1. Introducción

Muchos de los problemas que se presentan en el quehacer cotidiano de diversos sectores como el comercial y el empresarial, pueden formularse como un mAP, y en especial como un $3AP$, dadas las grandes áreas de aplicación, sus numerosas generalizaciones y problemas relacionados. Pero generalmente su planteamiento sugiere el desarrollo de un modelo cuya resolución resulta prácticamente intratable usando técnicas clásicas (entre las que se pueden mencionar: programación lineal, dinámica y simulación) pues siempre se resuelve haciendo simplificaciones razonables del problema original.

Entre los problemas que se pueden formular como un mAP, podemos mencionar: problemas de asignación de recursos, telecomunicaciones, operaciones de tripulación de aerolíneas, diseño eficaz de división de zonas escolares, entre otros.

Debido a que los métodos que emplean algoritmos exactos son incapaces de resolver muchos de los problemas que se plantean en la realidad, en el ámbito laboral existe la imperiosa necesidad de disponer de herramientas

que permitan obtener, de manera rápida, si no la óptima, una buena solución para el *mAP*.

Como se dijo en el capítulo anterior, el *mAP* es, en general, *NP*-difícil para $m \geq 3$. Por lo tanto, no existen algoritmos de resolución en tiempo polinomial. Esto último junto con su importancia en las operaciones prácticas, justifica la necesidad de un estudio profundo que conduzca a la propuesta y desarrollo de nuevas herramientas para su resolución.

Entre las herramientas existentes para la búsqueda de soluciones para el *mAP* se tienen: los algoritmos heurísticos y metaheurísticos, los algoritmos de relajación y algoritmos de ramificar y acotar.

2.2. Heurísticas y metaheurísticas

Definición 2.2.1 *Se llama algoritmo heurístico, metaheurístico o simplemente heurístico, a cualquier algoritmo que con poco esfuerzo computacional proporcione una solución factible cuyo valor objetivo, normalmente, debe estar próximo al valor objetivo óptimo del problema original.*

Este término, derivado de la palabra griega *heuriskein* que significa encontrar o descubrir, se usa en el ámbito de la optimización para describir una clase de algoritmos de resolución de problemas.

Los métodos heurísticos se limitan a proporcionar una buena solución no necesariamente óptima. Lógicamente, el tiempo invertido por un método exacto para encontrar la solución óptima de un problema *NP*-difícil, si es que existe tal método, es de un orden de magnitud muy superior al del heurístico (pudiendo llegar a ser tan grande en muchos casos, que sea inoperable).

En los últimos años ha habido un crecimiento espectacular en el desarrollo de procedimientos heurísticos para resolver problemas de optimización. Este

hecho queda claramente reflejado en el gran número de artículos publicados en revistas especializadas [Martí, 2002].

Existen muchas razones para la utilización de métodos heurísticos, además de ser utilizados para resolver problemas de tipo *NP*-difícil, entre las cuales podemos mencionar las siguientes:

. No existe un método exacto de resolución del problema.

. Aunque existe un método exacto para resolver el problema, éste requiere mucho tiempo de cálculo y memoria.

. No se necesita la solución óptima.

. Los datos son poco fiables.

. El método heurístico es más flexible que un método exacto, permitiendo por ejemplo, la incorporación de condiciones de difícil modelización.

. El método heurístico se utiliza como parte de un procedimiento global que garantiza el óptimo de un problema. Existen dos posibilidades:

  - El método heurístico proporciona una buena solución inicial de partida.

  - El método heurístico participa en un paso intermedio del procedimiento, como por ejemplo las reglas de selección de la variable a entrar en la base en el método simplex.

En los métodos heurísticos, las técnicas e ideas aplicadas a la resolución de un problema son específicas de éste y aunque, en general, pueden ser trasladadas a otros problemas, han de particularizarse en cada caso. Así pues, es necesario referirse a un problema concreto para estudiar con detalle los

procedimientos heurísticos. Es por ello que decimos que el estudio de los algoritmos heurísticos dependen en gran medida del problema concreto para el que se ha diseñado.

Existen muchos métodos heurísticos de naturaleza muy diferente. A continuación describimos cinco grandes métodos, no excluyentes, en donde ubicar los heurísticos más conocidos.

Métodos de descomposición. El problema original se descompone en subproblemas más pequeños y sencillos de resolver, teniendo en cuenta, almenos de manera general, que todos los subproblemas pertenecen al mismo problema.

Métodos inductivos. Aquí la idea es generalizar versiones pequeñas o más sencillas al caso completo. Propiedades o técnicas identificadas en estos casos más fáciles de analizar pueden ser aplicadas al problema completo.

Métodos de reducción. Consisten en identificar alguna característica que presumiblemente deba contener la solución óptima y de ese modo simplificar el problema. El objeto es restringir el espacio de soluciones. El riesgo, en estos casos, es dejar fuera las soluciones óptimas del problema original.

Métodos constructivos. Consisten en construir, literalmente, paso a paso una solución del problema, sin precisar de ninguna solución previa. Usualmente, son métodos deterministas y suelen estar basados en la mejor elección en cada iteración.

Métodos de búsqueda local. A diferencia de los anteriores, los procedimientos de búsqueda local o mejora local, también llamados de búsqueda por entornos, comienzan con una solución del problema y la mejoran progresivamente. El procedimiento realiza en cada paso un movimiento de una solución a otra con mejor valor. El método finaliza cuando, para una solución, no existe ninguna solución accesible que la mejore.

En los últimos años han aparecido una serie de métodos bajo el nombre de Metaheurísticos con el propósito de obtener mejores resultados que los alcanzados por los heurísticos tradicionales.

Cuando se habla de heurísticos se refiere a los métodos clásicos, pero cuando se habla de metaheurísticos nos estamos refiriendo a los métodos más recientes y complejos, es por ello que también se hace referencia a los metaheurísticos como "heurísticos modernos"[Reeves, 1995].

Los metaheurísticos que posteriormente analizaremos, pertenecen a la categoría de los métodos de búsqueda local. Es de hacer notar, que los métodos constructivos y los de búsqueda local constituyen la base de los procedimientos metaheurísticos.

Este trabajo se centra en métodos resultantes de combinar la construcción con la búsqueda local, puesto que pueden considerarse un punto de inicio en el desarrollo de métodos metaheurísticos.

Un algoritmo heurístico debe tener las siguientes propiedades:

1.- Eficiente. Un esfuerzo computacional realista para obtener la solución.

2.- Bueno. La solución debe estar, en promedio, cerca del óptimo.

3.- Robusto. La probabilidad de obtener una solución muy alejada del óptimo debe ser baja.

4.- Fácil de comprender e implementar.

5.- Eficiente. En cuanto al uso de los recursos disponibles, en especial tiempo y memoria.

Cuando se plantea cualquier nuevo procedimiento para resolver un problema, el investigador está interesado en analizar el comportamiento del procedimiento propuesto y en comparar éste con otras técnicas exactas o heurísticas,

conocidas o no, para resolver el mismo problema. Así, para medir el comportamiento o la calidad de un heurístico existen diversos procedimientos, entre los que se encuentran los siguientes:

Medición de la calidad de la solución propuesta por un heurístico

Aunque normalmente se recurre al heurístico por no existir un método exacto para obtener el óptimo, o por ser éste computacionalmente muy costoso, en ocasiones puede que dispongamos de un procedimiento que proporcione el óptimo para ejemplos de tamaño reducido. Estos ejemplos pueden servir para medir la calidad del método heurístico. Cuando esto es posible, se mide la desviación porcentual de la solución heurística frente a la óptima, y se calcula el promedio de dichas desviaciones. Sea c_h el coste de la solución heurística y c_{opt} el coste de la solución óptima de un ejemplo dado, en un problema de minimización la desviación porcentual, también llamada función *gap* viene dada por:

$$gap := \frac{c_h - c_{opt}}{c_{opt}} \times 100$$

Cuando la comparación directa con el óptimo no es posible, puede recurrirse a una comparación indirecta midiendo la distancia entre la solución heurística y cotas sobre las soluciones óptimas.

Uno de los métodos más empleados en problemas NP-difíciles es la comparación con otros heurísticos sobre los que se ha trabajado y se conocen algunos buenos resultados. Al igual que ocurre con la comparación con las cotas, la conclusión de dicha comparación está en función de la bondad del heurístico escogido.

Medición del esfuerzo computacional

Se está interesado en conocer, entre otros, el tiempo empleado hasta encontrar la mejor solución, el tiempo total de ejecución, el tiempo por fases y el tiempo de la relación esfuerzo-calidad. Este último se define como el ratio entre el tiempo necesario para hallar una solución dentro de un porcentaje especificado de la mejor solución encontrada y el tiempo necesario para encontrar esta última.

Un ejemplo de heurísticas constructivas son los llamados algoritmos *greedy*, cuya lógica es que si se quiere alcanzar cierta meta en varias etapas sucesivas, en cada una de ellas debe tratar de hacer lo mejor que se pueda hacer en ese momento, y dicha decisión no se cambia tras decisiones futuras.

En los algoritmos *greedy* se define una función *greedy* que mide el beneficio de añadir cada uno de los elementos según la función objetivo y elegir la mejor. Nótese que esta medida es miope en el sentido que no tiene en cuenta qué ocurrirá en iteraciones sucesivas al realizar una elección, sino únicamente en esa iteración.

Un algoritmo *greedy* puede describirse de manera general como se muestra en la Figura 2.1.

En los últimos años, en la búsqueda de mejores resultados, han aparecido una serie de métodos bajo el nombre de metaheurísticos, con el fin de obtener mejores resultados que los alcanzados por los heurísticos tradicionales.

El término metaheurístico fue introducido por Fred Glover en 1986 y los profesores Osman y Kelly, en 1995, introducen la siguiente definición [Martí, 2002].

Definición 2.2.2 *Los procedimientos metaheurísticos son una clase de métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria, en los que los heurísticos clásicos no son efectivos.*

```
Procedure GREEDY
    main {
        S = ∅ ;
        while (existe e ∉ S : z(S ∪ {e}) < z(S)) {
            elegir tal e;
            S = S ∪ {e};
            }
    }
```

Figura 2.1: Algoritmo *greedy* general.

Los metaheurísticos proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la Inteligencia Artificial, la evolución biológica y los mecanismos estadísticos.

Entre los algoritmos metaheurísticos más conocidos se encuentran: búsqueda tabú, métodos evolutivos (algoritmos genéticos, búsqueda dispersa y sistema hormiga), templado simulado y métodos GRASP, los cuales están dentro de la categoría de los métodos de búsqueda local.

En este trabajo desarrollamos algoritmos utilizando las metaheurísticas búsqueda tabú, algoritmos genéticos y sistema hormiga, combinándolas con unos algoritmos *greedy*.

2.3. Métodos evolutivos

Los métodos evolutivos están basados en poblaciones de soluciones. En cada iteración del algoritmo no se tiene una única solución, sino un con-

junto de éstas. Estos métodos se basan en generar, seleccionar, combinar y reemplazar un conjunto de soluciones. Dado que mantienen y manipulan un conjunto en lugar de una única solución a lo largo de todo el proceso de búsqueda, suelen presentar tiempos de computo más altos que otros metaheurísticos.

Entre los métodos evolutivos se encuentran los algoritmos genéticos, búsqueda dispersa y sistema hormiga.

2.3.1. Algoritmos genéticos

En la naturaleza, la Evolución de los seres vivos tiene algunas características que motivaron a John Holland a comenzar una línea de investigación. Con el fin de emular a la Naturaleza, que ha logrado que sus organismos evolucionen bajo diversas condiciones ambientales, adaptándose a cambios y haciendo que a través de generaciones se especialicen para sobrevivir (según lo planteado por Darwin), nace esta línea de investigación que eventualmente se transformó en lo que hoy en día se denomina *Algoritmos Genéticos (AGs)* [Díaz et al., 1996].

Los *AGs* fueron introducidos por Holland en 1970 inspirándose en el proceso observado en la evolución natural de los seres vivos. De manera general podemos decir que en la evolución de los seres vivos el problema al que cada individuo se enfrenta cada día es la supervivencia. Para ello cuenta con las habilidades innatas provistas en su material genético.

En general, la evolución se produce en dos procesos: la selección y la alteración genética de los cromosomas que almacenan la característica de la especie.

Aunque muchos aspectos, en la evolución natural de los seres vivos están todavía por discernir, existen unos principios generales de la evolución bi-

ológica ampliamente aceptados por la comunidad científica. Algunos de estos son:

. La evolución opera en los *cromosomas* en lugar de en los individuos a los que representa.

. La selección natural es el proceso por el que los cromosomas con "buenas estructuras" se reproducen más a menudo que los demás.

. En el proceso de reproducción tiene lugar la evolución mediante la combinación de los cromosomas de los progenitores. Llamamos *recombinación* a este proceso en el que se forma el cromosoma del descendiente.

. Las *mutaciones* pueden causar que los cromosomas de los hijos sean diferentes a los de los padres y los procesos de recombinación pueden crear cromosomas bastante diferentes en los hijos, por la combinación de material genético de los cromosomas de los padres.

. La evolución biológica *no tiene memoria*, en el sentido de que en la formación de los cromosomas únicamente se considera la información del período anterior.

En la naturaleza, los anteriores procesos ocurren sobre una generación, y luego sobre su descendencia, y a continuación sobre la descendencia de ésta, y así sucesivamente. Después de cada ciclo, la generación actual será (al menos así se desea) mejor que las anteriores. Esto es, los individuos estarán más evolucionados y adaptados al medio [Moreno y Moreno, 1999].

La habilidad de una población de cromosomas para explorar el espacio de búsqueda "en paralelo" y combinar lo mejor que ha sido encontrado en

él mediante el mecanismo de recombinación, es algo intrínseco a la evolución natural y trata de ser explotado por los *AGs*.

El campo de los *AGs* ha evolucionado, principalmente debido a las innovaciones introducidas en la década de 1980. Estos han ido incorporando mecanismos cada vez más elaborados, bajo la motivación de la necesidad de resolución, aproximada, de problemas prácticos de amplia variedad [Díaz et al., 1996].

Los *AGs* establecen una analogía entre el conjunto de soluciones de un problema y el conjunto de individuos de una población natural, codificando la información de cada solución en una cadena de símbolos, por ejemplo un vector binario, a modo de cromosoma.

A tal efecto se introduce una función de evaluación de los cromosomas que llamaremos calidad ("*fitness*") y que está basada en la función objetivo del problema. Igualmente, se introduce un mecanismo de selección de manera que los cromosomas con mejor evaluación sean escogidos para "reproducirse" más a menudo que aquellos cuya evaluación sea peor.

Los *AGs* están basados en integrar e implementar eficientemente dos ideas fundamentales: las representaciones simples como vectores binarios de las soluciones del problema y la realización de transformaciones simples para modificar estas representaciones.

Para utilizar las ventajas del proceso evolutivo de la naturaleza en la resolución de un problema de optimización, deben hacerse las siguientes consideraciones [Moreno y Moreno, 1999]:

. Una apropiada codificación de las soluciones del problema representará a estos de la misma forma que el cromosoma representa a los individuos de la especie. Dada esta unívoca relación, a partir de ahora se usarán indistintamente los términos solución, codificación, cromosoma,

individuo.

. La adecuación de cada solución será una medida del comportamiento de ésta en el problema considerado. Normalmente, se considera que es sólo función del valor objetivo de la solución, pero pueden tenerse en cuenta otros elementos como son la robustez de la solución, el recurso necesario para obtener una mejor solución, entre otros.

. Se definirán unos operadores genéticos que, al actuar sobre una variable o varias soluciones, suministren una o más soluciones al alterar genéticamente los cromosomas.

. Una población inicial que será la base a partir de la cual evolucionará la solución mediante la aplicación de los operadores genéticos.

Para la representación genética de las soluciones del problema, se consideran las entradas y salidas del ambiente donde evoluciona la población. Éstas pueden ser representadas con una cadena de símbolos de longitud finita (codificación) basados en un *alfabeto* dado, el cual es un conjunto finito de símbolos.

Para escoger el alfabeto de los *AGs* se basan en el principio del mínimo alfabeto que expresa lo siguiente:

"El diseñador de un AG deberá seleccionar el alfabeto más pequeño que permita expresar de manera natural el problema"

El alfabeto que representa al problema debe ser claro, de tal manera que el espacio de búsqueda pueda ser acotado al espacio exacto del problema.

Definición 2.3.1 *Se llama individuo a cada punto del espacio del problema, el cual es representado unívocamente por una cadena de caracteres que se genera a partir del alfabeto seleccionado.*

Definición 2.3.2 *Un cromosoma es una cadena de símbolos, que contienen símbolos únicos (genes) que toman valores (alelos), sobre los cuales cada individuo sirve como material genético con una posición específica.*

Definición 2.3.3 *Un operador genético es aquel que, al actuar sobre una o varias soluciones, suministra una o más soluciones al alterar genéticamente los cromosomas.*

En los trabajos realizados inicialmente sobre o utilizando AGs, el alfabeto utilizado era el sistema binario, es decir, listas de 1s y 0s. Este tipo de representación ha sido ampliamente utilizada incluso en problemas donde no es natural.

En la actualidad, podemos distinguir dos escuelas. La primera se limita al uso del sistema binario como alfabeto, mientras que la segunda utiliza todo tipo de configuración. Hemos de notar que las operaciones genéticas dependen del tipo de representación, por lo que la elección de una condiciona la otra [Martí, 2002].

La población inicial suele ser generada aleatoriamente. Sin embargo, últimamente se están utilizando métodos heurísticos para generar soluciones iniciales de calidad. En este caso, es importante garantizar la diversidad estructural de estas soluciones para tener una representación de la mayor parte de la población posible o al menos evitar la convergencia prematura.

Respecto a la evaluación de los cromosomas, se suele utilizar la calidad como medida de bondad según el valor de la función objetivo en el que se puede añadir un factor de penalización para controlar la infactibilidad. Este factor puede ser estático o ajustarse dinámicamente.

Los operadores genéticos son los encargados de ejecutar acciones sobre los individuos de la población actual $P(t)$ (padres), para obtener una nueva generación $P(t+1)$ (hijos) con individuos soluciones cada vez más aptos.

Los operadores genéticos estándar son: operador selección, operador cruce y operador mutación.

. **Operador selección**. Dada una población de individuos $P(t)$, los individuos que conforman la próxima generación son escogidos de $P(t)$ mediante un proceso de generación probabilística, la cual asegurará que el número de veces que se espera que un individuo sea seleccionado es proporcional al rendimiento relativo del individuo en comparación con el resto de la población, así el "mejor" cromosoma hará más veces de padre que el "peor".

. **Operador cruce**. Éste combina los caracteres de dos individuos (padres) para formar descendientes hijos con carga genética similar, y actúa intercambiando segmentos o puntos entre las representaciones de los individuos, produciendo dos nuevos individuos para el espacio solución del problema. Los operadores cruce más utilizados son:

  - **De un punto**. Se elige aleatoriamente un punto de ruptura en los padres y se intercambian sus genes. Como se muestra en la Figura 2.2.
  - **De dos puntos**. Se eligen dos puntos de ruptura al azar para intercambiar los genes. Como se muestra en la Figura 2.3.
  - **Uniforme**. En cada gen se elige un padre para que contribuya con su gen al del hijo, mientras que el segundo hijo recibe el gen del otro padre. Como se muestra en la Figura 2.4.

    En la Figura 2.4 se realiza la elección que se describe a continuación.

    Para el hijo 1 se seleccionaron los siguientes genes:

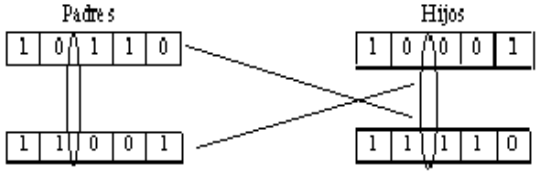

Figura 2.2: Ejemplo de cruce de un punto.

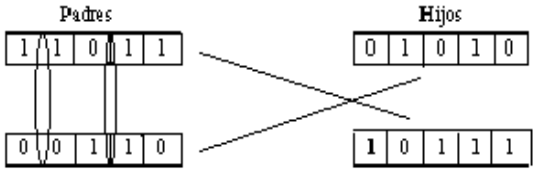

Figura 2.3: Ejemplo de cruce de dos puntos.

Gen 1 - padre 1 $\rightarrow$ 1.

Gen 2 - padre 2 $\rightarrow$ 1.

Gen 3 - padre 2 $\rightarrow$ 0.

Gen 4 - padre 1 $\rightarrow$ 1.

Gen 5 - padre 1 $\rightarrow$ 0,

por lo tanto al hijo 2 se le asignaron los siguientes genes:

Gen 1 - padre 2 $\rightarrow$ 0.

Gen 2 - padre 1 $\rightarrow$ 0.

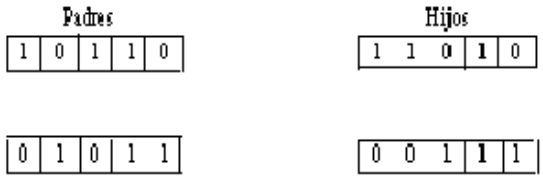

Figura 2.4: Ejemplo de cruce uniforme.

Gen 3 - padre 1 $\rightarrow$ 1.

Gen 4 - padre 2 $\rightarrow$ 1.

Gen 5 - padre 2 $\rightarrow$ 1.

- **PMX, SEX**. Son operadores más sofisticados producto de mezclar y aleatorizar los anteriores.

. **Operador mutación**. La operación mutación más sencilla, y una de las más utilizadas consiste en reemplazar con cierta probabilidad el valor de un gen. Este operador también actúa aleatoriamente sobre los individuos de una generación $P(T)$ alterando arbitrariamente uno o más valores de los genes de la estructura seleccionada, este operador incrementa la variabilidad de la población, ya que aumenta las posibilidades de ampliar el espacio de búsqueda a zonas que de otra manera quedarían ocultas.

Un algoritmo genético estándar [Vignaux y Michalewicz, 1991] está descrito de modo tal que se pueden determinar las acciones que se deben realizar para alcanzar la evolución de la población. Éste se presenta en la Figura 2.5.

Descripción del procedimiento. Durante la t-ésima iteración, se dispone de una población $P(t) := \{x_1^t, ..., x_h^t\}$, de soluciones. Cada solución se evalúa

```
Procedure GENÉTICO
    main {
        t := 0 ;
        inicializar P(t);
        evaluar P(t);
        while (condición de parada) {
            t := t + 1;
            seleccionar P(t) desde P(t − 1);
            operar P(t);
            evaluar P(t);
        }
    }
```

Figura 2.5: Algoritmo genético estándar.

para tener una medida de su adecuación. La población de la etapa $(t+1)$ se forma seleccionando, en primer lugar, los individuos más adecuados de $P(t)$. Después de esta fase de selección, se opera genéticamente sobre las codificaciones. Estos pasos se reiteran hasta que se cumpla el criterio de parada.

Convergencia del algoritmo. Dado que los AGs operan con una población en cada iteración, se espera que el método converja de modo que al final del proceso la población sea muy similar, y en el infinito se reduzca a un solo individuo.

Se ha desarrollado toda una teoría para estudiar la convergencia de estos algoritmos en el caso de usar codificación binaria. Esta teoría se basa principalmente en considerar que un vector binario es una representación de una clase de equivalencia o esquema reinterpretando la búsqueda en lugar de entre vectores binarios entre esquemas.

En el caso de un vector no binario, se introducen los conceptos de forma y conjunto de similitud que generalizan el de esquema. Se consideran una serie de condiciones sobre los operadores de manera que se garantice la convergencia. Básicamente, se exige que al cruzar dos cromosomas de la misma clase se obtenga otro de ésta. Además, hay que respetar ciertas condiciones sobre selección de los progenitores. Bajo toda esta serie de hipótesis se prueba la convergencia del algoritmo.

En la práctica no se suelen respetar estas condiciones, ya que son difíciles de seguir y probar, encontrándose con que en ocasiones los *AGs* resuelven satisfactoriamente un problema de optimización dado y en otras se quedan muy alejados del óptimo.

Greenhalgh y Marshall (2000), observando el efecto de la mutación en la convergencia, mostraron la convergencia de un algoritmo genético ejecutando éste durante un tiempo suficientemente largo para garantizar la convergencia a un óptimo global, con cualquier específico nivel de confianza. Ellos obtienen una cota superior para el número de iteraciones necesarias para asegurar la mejora de resultados anteriores. Las cotas superiores obtenidas decrecen inversamente proporcional al tamaño de la población.

Es importante citar que, a diferencia de otros metaheurísticos, los *AGs* han crecido de forma espectacular, hasta el punto de poder encontrar referencias en revistas de informática de carácter general. Además, muchos de los investigadores de este campo están trabajando en desarrollar los aspectos teóricos, e incorporando algunas otras técnicas de búsqueda local en el esquema genético.

Los Algoritmos Meméticos son un caso particular de estos nuevos híbridos que aúnan los elementos de la combinación con los de búsqueda local bajo el nombre de cooperación y competición.

2.3.2. Búsqueda dispersa

La Búsqueda Dispersa *(BD)* es un método evolutivo que ha sido aplicado en la resolución de un gran número de problemas de optimización.

La primera descripción del método fue publicada en 1977 por Fred Glover, donde establece los principios de la *BD* determinando que ésta realiza una exploración sistemática sobre una serie de buenas soluciones llamadas conjunto de referencias [Martí, 2002].

Al igual que los *AGs*, este método se basa en mantener un conjunto de soluciones y realizar combinaciones con éstas; pero a diferencia a ellos no está fundamentado en la aleatorización sobre un conjunto relativamente grande de soluciones, sino en las elecciones sistemáticas y estratégicas sobre un conjunto pequeño de éstas [Martí, 2002].

Glover, Laguna y Martí (2000), estudian las implementaciones más recientes del método en la resolución de problemas de optimización combinatoria. Además, describen los rasgos de *BD* y su generalización llamada re-encadenamiento de trayectorias, que los colocan aparte de otros métodos evolutivos y que ofrecen oportunidad para mejores y más métodos efectivos y versátiles en el futuro.

El algoritmo de *BD*. El método se basa en combinar las soluciones que aparecen en el llamado conjunto de referencia. En este conjunto se tienen las soluciones "buenas" que se han ido encontrando. Es importante destacar que "buena" no se restringe a la calidad de la solución, sino que también se considera la diversidad que ésta aporta al conjunto. La *BD* consta básicamente de los siguientes elementos:

. **Un generador de soluciones diversas**. El método se basa en generar un conjunto P de soluciones diversas (alrededor de 100), del que se extrae un subconjunto pequeño (alrededor de $b := 10$) con el que se

realizan las combinaciones y que se denomina R.

. **Un conjunto de referencia** R. Extraído del conjunto de soluciones diversas. Estas soluciones están ordenadas de "mejor" a "peor" respecto a su calidad.

  - **Creación**. Se inicia R con las $b/2$ mejores soluciones de P. Las $b/2$ restantes se extraen de P por el criterio de máxima distancia con las ya incluidas en R. Para ello se define previamente una función distancia en el problema.
  - **Actualización**. Los resultados de las combinaciones pueden entrar en R y reemplazar a aquéllas incluidas que sean mejoradas.

. **Un método de combinación**. *BD* se basa en combinar todas las soluciones en R, considerando subconjuntos de 2 o más elementos y combinándolos mediante una rutina diseñada a tal efecto. Estas soluciones obtenidas pueden ser inmediatamente introducidas en R (actualización dinámica) o almacenada temporalmente y después ver que soluciones entran a R (actualización estática).

. **Un método de mejora**. Un método de búsqueda local para mejorar las soluciones, tanto de R como de las combinaciones, antes de estudiar su inclusión en R.

En la Figura 2.6 presentamos un esquema básico del algoritmo de *BD* [Martí, 2002].

El algoritmo hace referencia a los subconjuntos de R, ya que se pueden combinar pares, tríos o cualquier número de soluciones y se detiene cuando al tratar de combinarlos no hay nuevos elementos en R.

Procedure BÚSQUEDA DISPERSA

main {

$P := \varnothing$;

while ($|P| < 100$){

Generar una solución y mejorarla;

Sea x solución mejorada. Si $x \notin P$, hacer $P := P \cup \{x\}$;

Sea $R := \{x^1, ..., x^h\} \subset P$ con $|R| := b/2$ las $b/2$ mejores soluciones de P;

}

Evaluar $x^i \in R$, $i := 1, ..., h$ y ordenarlas de mejor

a peor respecto a la función objetivo;

$NuevaSolucion := TRUE$;

while ($NuevaSolucion$) {

$NuevaSolucion := FALSE$;

Generar $S \subset R$, donde S tiene al menos una nueva solución x^j;

while(Subconjuntos sin examinar) {

Seleccionar $S \subset R$ y etiquetarlo como examinado;

Aplicar método de combinación a las soluciones de S y mejorar

las soluciones obtenidas;

Sea x solución mejorada;

If $f(x)$ mejora a $f(x^b)$ y $x \notin R$ {

$x^b := x$ y reordenar R;

$NuevaSolucion := TRUE$;

}

}

}

}

Figura 2.6: Algoritmo de *BD*.

2.3.3. Sistema hormiga

En una colonia de hormigas la estrategia empleada para descubrir fuentes de alimentación consiste en: establecer el camino más corto entre estos y el hormiguero, y transmitir esta información al resto de la colonia. Este hecho inspiró a los investigadores Dorigo, Maniezzo y Colorni (1991), a emular dicha estrategia proponiendo un procedimiento de solución para problemas combinatorios.

A la hora de buscar comida, inicialmente, las hormigas desarrollan una búsqueda aleatoria alrededor del hormiguero. Si alguna de ellas encuentra una fuente de alimentación, transporta cierta cantidad de alimento al hormiguero y deja en su recorrido un rastro de *feromona* (sustancia química que excretan algunos animales y que influye en el comportamiento de otros de su especie). Este rastro sirve de guía a otras hormigas que, al realizar la misma operación, marcan con su feromona el camino volviéndose cada vez más intenso el rastro. Ahora, si la fuente se vuelve poco apetecible para las hormigas, éstas realizan el recorrido cada vez en menor cantidad y frecuencia, lo que hace que el rastro tienda a desaparecer por efectos de la evaporación.

En caso de que entre la fuente y el hormiguero se encuentre un obstáculo que no pueda atravesarse, las hormigas bordean el mismo tomando al azar la ruta a seguir (izquierda o derecha). Al igual que anteriormente, por efectos de la evaporación, el rastro dejado sobre el camino más largo, es más débil y tiende a desaparecer más rápidamente, que el dejado en el camino más corto, estableciéndose nuevamente el camino más corto entre ambos puntos [Moreno y Moreno, 1999].

De acuerdo a lo anteriormente descrito, se tiene que las hormigas construyen iterativamente soluciones al problema que se les plantea e intercambian información sobre éstas para construir mejores soluciones.

Para las hormigas, la atracción que sienten por un determinado camino es proporcional a la intensidad del rastro de feromona sobre el mismo. Esta idea puede aplicarse a la resolución de un problema si se supone que un agente (hormiga artificial), al construir una solución, asocia a cada elemento constituyente de la misma un rastro proporcional a la calidad de aquella.

De la capacidad que tienen las hormigas para establecer el camino más corto desde el hormiguero a la fuente de alimentación y viceversa, como se ha expresado con anterioridad, ha surgido una técnica de optimización llamada *Sistema Hormiga ("Ant System", AS)*, la cual está enmarcada dentro de las llamadas metaheurísticas.

Como el procedimiento heurístico lo que trata es de emular el proceso natural y no de imitarlo fielmente, éste puede mejorarse utilizando información adicional. Luego, se puede disponer de dos medidas para realizar una determinada elección: la que aporta la intensidad de feromona asociada a esa elección y la que se obtiene a través de una evaluación heurística de los movimientos permitidos de tal elección. Estas dos medidas deben usarse para guiar el proceso de búsqueda de buenas soluciones.

En un algoritmo *AS* se debe especificar: cómo se inicializa el rastro, cómo se obtiene su incremento, cómo se actualiza, cómo se elige el próximo elemento de la lista y cuál es el criterio de parada, sin olvidar la heurística que se emplea en combinación con la intensidad del rastro para decidir cuál elemento de la lista se escoge.

2.4. Búsqueda tabú

Los algoritmos de *Búsqueda Tabú* (*TS* por su nombre en inglés, *"Tabu Search"*) fueron introducidos, oficilamente, por Glover (1989), como una

meteheurística diseñada para obtener buenas soluciones a problemas de optimización. Sin embargo, los orígenes de *TS* pueden situarse en diversos trabajos publicados a finales de los años 70 [Moreno y Moreno, 1999].

Definición 2.4.1 *Cuando la búsqueda de buenas soluciones se desarrolla en los alrededores de una solución actual, se denomina búsqueda local.*

Definición 2.4.2 *Búsqueda tabú (TS), es un procedimiento metaheurístico utilizado para guiar un algoritmo heurístico de búsqueda local para explorar el espacio de soluciones más allá de la simple optimalidad local y está basada en principios generales de Inteligencia Artificial (IA).*

TS pertenece a una clase de métodos heurísticos de búsqueda local, o más específicamente a una clase de Métodos Descendentes-Ascendentes [Hansen y Jaumard, 1990]. Ésta se basa en la premisa que para poder calificar de inteligente la solución de un problema, debe incorporar memoria adaptativa y exploración sensible, para ello toma de la *IA* el concepto de memoria y lo implementa mediante estructuras simples, con el objeto de dirigir la búsqueda teniendo en cuenta la historia de ésta.

Definición 2.4.3 *Un movimiento es una operación que permite obtener una solución desde otra.*

Definición 2.4.4 *El conjunto de soluciones que se obtienen desde una dada, empleando un movimiento particular, constituyen el entorno o vecindad de la solución asociada a tal movimiento.*

Definición 2.4.5 *Una estructura de entorno, para un problema particular, es una aplicación que asocia a cada solución factible un conjunto de soluciones que están cercanas, en algún sentido, a la solución dada.*

Un entorno o vecindad de una solución **x** de un problema dado se denota por $N(\mathbf{x})$.

TS o híbridos de *TS* con otras heurísticas o procedimientos algorítmicos, han sido utilizados para encontrar mejores soluciones a problemas en programación, secuenciación, asignación de recursos, planificación de inversiones, telecomunicaciones y en muchas otras áreas.

TS comienza de la misma forma que cualquier procedimiento de búsqueda local, procediendo iterativamente de una solución x a otra y en el entorno de la primera: $N(x)$. Sin embargo, en lugar de considerar todo el entorno de una solución, *TS* define el entorno reducido $N^*(x) \subseteq N(x)$ como aquellas soluciones disponibles en $N(x)$, las cuales serán las únicas soluciones alcanzables a partir de x.

Definición 2.4.6 *Las soluciones que no son admitidas en $N^*(x)$ se denominan tabú.*

Definición 2.4.7 *Las soluciones tabú son almacenadas en una lista durante cierto número de iteraciones, la cual se denomina lista tabú, cuyo tamaño se puede ajustar dinámicamente según la estrategia utilizada.*

Existen varias formas de definir el entorno reducido de una solución. La más sencilla consiste en etiquetar como tabú las soluciones previamente visitadas en un pasado cercano. Esta forma se conoce como memoria a corto plazo (*short term memory*) y está basada en una lista tabú T de las soluciones visitadas recientemente (*Recency*). Así, en una iteración determinada se tiene que $N^*(x) := N(x) - T$ [Martí, 2002][Díaz et al., 1996].

El objetivo principal de etiquetar las soluciones visitadas como tabú es el de evitar que la búsqueda se cicle. Luego de un cierto número de iteraciones,

la búsqueda estará en una región distinta y podrán liberarse estas soluciones del status tabú, reduciendo así el esfuerzo computacional [Martí, 2002].

Definición 2.4.8 *Se define nivel de aspiración a aquellas condiciones que, de satisfacerse, permitirán alcanzar una solución aunque tenga status tabú.*

Una implementación sencilla del nivel de aspiración consiste en permitir alcanzar una solución siempre que mejore a la mejor almacenada, aunque sea tabú.

Es importante considerar que los métodos basados en búsqueda local requieren de la exploración de un gran número de soluciones en poco tiempo, por ello es importante reducir al mínimo el esfuerzo computacional de las operaciones que se realizan a menudo.

La memoria usada en *TS* puede ser explícita o basada en atributos, aunque ambas modalidades no son excluyentes. La memoria explícita conserva soluciones completas, y consiste en una élite de soluciones visitadas durante la búsqueda (o entornos altamente atractivos pero inexplorados por tales soluciones). Estas soluciones especiales se introducen estratégicamente para ampliar $N^*(x)$, y así presentar opciones útiles que no se encuentran en $N(x)$ [Martí, 2002][Díaz et al., 1996]..

La memoria basada en atributos produce un efecto más sutil en la búsqueda, ésta guarda información sobre atributos de las soluciones que cambian al moverse de una solución a otra. Por ejemplo, en un contexto de grafos o redes, los atributos pueden consistir de nodos o arcos que se añaden, se suprimen o sustituyen por los movimientos ejecutados. Así, un atributo que fue ejecutado como tabú por pertenecer a una solución visitada hace n iteraciones, puede impedir en la iteración actual alcanzar una solución por contenerlo, aunque ésta sea diferente de la que provocó el que el atributo fuese etiquetado. Esto

permite, a largo plazo, el que se identifiquen y mantengan aquellos atributos que inducen una cierta estructura beneficiosa en las soluciones visitadas.

Un algoritmo *TS* está basado en la interacción entre la memoria a corto plazo y la memoria a largo plazo. Ambos tipos de memoria llevan asociadas sus propias estrategias y atributos, y actúan en ámbitos diferentes.

2.4.1. Memoria a corto plazo y sus elementos

Definición 2.4.9 *La memoria a corto plazo que lleva la cuenta de los atributos de la solución que han sido cambiados en el pasado reciente es llamada memoria basada en recencia (soluciones visitadas en el pasado reciente).*

Es de hacer notar, que la memoria basada en recencia es la memoria a corto plazo más comúnmente usada.

Para explorar esta memoria, los atributos seleccionados que se presentan en soluciones recientemente visitadas son designados como "tabú-activos", y las soluciones que contienen elementos tabú-activos, o combinaciones particulares de estos atributos, son las que se convierten en tabú. Esto evita que algunas soluciones del pasado reciente pertenezcan a $N^*(x)$ y por tanto que sean revisitadas.

A continuación describimos los elementos de la memoria a corto plazo en la búsqueda tabú.

Manejo de memoria basada en recencia. El proceso se maneja creando una o más listas tabú, las cuales registran los atributos tabú-activo y explícita o implícitamente identifican su status actual.

Definición 2.4.10 *Se denomina tenencia tabú, a la duración, medida en número de iteraciones, que un atributo permanece tabú-activo.*

La tenencia tabú puede variar para diferentes tipos o combinaciones de atributos, y además puede variar para diferentes intervalos de tiempo o etapas de la búsqueda, permitiendo la creación de diferentes tipos de ajuste entre estrategia a corto y largo plazo.

Niveles de aspiración. El criterio de aspiración produce un elemento importante de flexibilidad en *TS*. El status tabú de una solución (o un movimiento) puede ser ignorado si ciertas condiciones se cumplen, ya que los niveles de aspiración dan umbrales de atracción, los cuales controlan el hecho de que las aspiraciones pueden ser consideradas admisibles a pesar de estar clasificadas como tabú.

Estrategias para la lista de candidatos. Las estrategias para la lista de candidatos son usadas para restringir el número de soluciones examinadas en una iteración dada, para los casos en los que $N^*(x)$ es grande o la evaluación de sus elementos es costosa.

En *TS* es importante contar con reglas eficientes para la generación y evaluación de buenos candidatos. Aún en casos donde las estrategias para la lista de candidatos no se usan explícitamente, las estructuras de memoria que den actualizaciones eficientes de evaluaciones del movimiento de una iteración a otra, y que reduzcan el esfuerzo de encontrar mejores o casi mejores movimientos, son parte integral de las implementaciones de *TS*. La actualización inteligente puede reducir apreciablemente los tiempos de solución, y la inclusión de estrategias para la lista de candidatos, para problemas grandes pueden aumentar significativamente los beneficios resultantes.

2.4.2. Memoria a largo plazo

En algunas aplicaciones, los componentes de la memoria *TS* de corto plazo son suficientes para producir soluciones de muy alta calidad. Sin em-

bargo, en general, *TS* se vuelve significativamente más potente incluyendo memoria a largo plazo y sus estrategias asociadas.

Definición 2.4.11 *Se denomina memoria a largo plazo, a aquella que almacena las frecuencias u ocurrencias de atributos en las soluciones visitadas tratando de identificar o diferenciar regiones.*

Tipos especiales de memoria basada en frecuencia son fundamentales en consideraciones de largo plazo. Éstas operan introduciendo penalizaciones e incentivos determinados por el rango relativo de tiempo durante el que los atributos han pertenecido a soluciones visitadas durante la búsqueda, permitiendo diferenciación por regiones. Las frecuencias de transición mantienen un registro de con qué frecuencia cambian los atributos, mientras que las frecuencias de residencia mantienen el registro de las duraciones relativas de los atributos en las soluciones generadas.

Es posible que el uso de memoria a largo plazo no requiera secuencias de larga duración antes de que sus beneficios se hagan visibles. Frecuentemente, sus mejoras comienzan a manifestarse en lapso de tiempo relativamente corto, y puede permitir que los esfuerzos para llegar a la solución finalicen antes que de alguna otra manera posible, debido a que son encontradas soluciones de alta calidad dentro de un rango corto de tiempo.

La memoria a largo plazo tiene dos estrategias asociadas las cuales describimos a continuación.

Estrategias de intensificación. La idea detrás del concepto de intensificación de la búsqueda es que, como un humano inteligente lo haría probablemente, se deben explorar más completamente las porciones del espacio de búsqueda que parecen "promisorias"para asegurarse que las mejores soluciones en estas áreas son halladas. Entonces, se puede decir que la intensificación consiste en regresar a regiones ya exploradas para estudiarlas más

a fondo. Para ello se favorece la aparición de aquellos atributos asociados a buenas condiciones encontradas. De vez en cuando, se detendría el proceso de búsqueda normal para realizar una fase de intensificación [Gendreau, 2003].

Dos variantes han demostrado tener bastante éxito. La primera introduce una medida de diversificación para asegurar que las soluciones registradas difieren una de otra en un grado deseado, y borra toda memoria de corto plazo antes de reanudar el proceso desde la mejor de las soluciones registradas. La otra variante, mantiene una lista secuencial de longitud limitada que añade al final una nueva solución sólo si es mejor que cualquier otra previa vista [Díaz et al., 1996].

El actual y último miembro de la lista es siempre el escogido (y suprimido) como base para reanudar la búsqueda. Sin embargo, la memoria a corto plazo *TS* que acompañó a esta solución también es guardada, y el primer movimiento inhibe además el movimiento previamente tomado de esta solución, con lo que un nuevo camino de solución será tomado.

Otro tipo de enfoque de intensificación es la "intensificación por descomposición", donde se pueden imponer restricciones a partir del problema o a la estructura de solución para generar una forma de descomposición que permita un enfoque más concentrado en otras partes de la estructura [Díaz et al., 1996].

La intensificación por descomposición también incluye otro tipo de consideraciones estratégicas, como basar la descomposición no sólo en indicadores de fuerza y consistencia, sino también en las oportunidades de elementos particulares para interactuar productivamente.

La intensificación es usada en muchas aplicaciones de *TS*, pero no siempre es necesario. Esto es porque hay muchas situaciones donde el buscador por el proceso normal de búsqueda está bastante completo. No hay necesidad

de pasarse tiempo explorando más cuidadosamente las porciones del espacio de búsqueda visitadas, y este tiempo puede usarse más efectivamente como veremos a continuación.

Estrategias de diversificación. La diversificación consiste en visitar nuevas áreas no exploradas del espacio de soluciones. Para ello se modifican las reglas de elección para incorporar a las soluciones atributos que no han sido usados frecuentemente. Alternativamente, se pueden introducir dichos atributos al reiniciar parcial o completamente el proceso de solución.

Las estrategias de diversificación pueden utilizar también una forma de largo plazo de memoria basada en recencia, cuyos resultados incrementan la tenencia tabú de los atributos de la solución.

Existen otros elementos más sofisticados de *TS* que, aunque poco probados, han dado muy buenos resultados en algunos problemas. Entre ellos se pueden destacar:

Movimientos de influencia. Son aquellos movimientos que producen un cambio importante en la estructura de las soluciones. Usualmente, en un movimiento de búsqueda local, la búsqueda es dirigida mediante la evaluación de la función objetivo. Su utilidad principal es la determinación de estructuras subyacentes en las soluciones. Esto permite que sean la base para procesos de Intensificación y Diversificación.

Oscilación estratégica. Ésta está estrechamente ligada a los orígenes de búsqueda tabú, y proporciona un medio para lograr una interacción efectiva entre intensificación y diversificación, ya sea a mediano o largo plazo. La oscilación estratégica opera orientando los movimientos en relación a una cierta frontera, donde el método se detendrá normalmente. Sin embargo, en vez de detenerse, las reglas para la elección de los movimientos se modifican para permitir que la región del otro lado de la frontera sea alcanzada; para

luego regresar a la zona inicial. El proceso de aproximarse, traspasar y volver sobre una determinada frontera crea un patrón de oscilación que da nombre a esta técnica. Una implementación sencilla consiste en considerar la barrera de la factibilidad/infactibilidad de un problema dado.

Elecciones probabilísticas. Normalmente *TS* se basa en reglas sistemáticas en lugar de decisiones al azar. Sin embargo, en ocasiones se recomienda el aleatorizar algunos procesos para facilitar la elección de buenos candidatos o cuando no está clara la estrategia a seguir. La selección aleatoria puede ser uniforme y seguir una distribución de probabilidad construida empíricamente a partir de la evaluación asociada a cada movimiento.

Umbrales tabú. Este procedimiento, conocido como *tabu thresholding (TT)*, se propone para aunar ideas que provienen de la oscilación estratégica y de las estrategias de lista de candidatos en un marco sencillo que facilite su implementación. El uso de la memoria es implícito en el sentido que no hay una lista tabú en donde anotar el status de los movimientos, pero la estrategia de elección de los mismos previene el ciclado. *TT* utiliza elecciones probabilísticas y umbrales en las listas de candidatos para implementar los principios de *TS*.

Encadenamiento de trayectorias. Este método conocido como *Path Relinking* (*PR*) [Glover, 1989][Glover, 1995], proporciona una integración muy útil de las estrategias de intensificación y diversificación, y se basa en volver a unir dos buenas soluciones mediante un nuevo camino. Así, si en el proceso de búsqueda se han encontrado dos soluciones x e y con un buen valor de la función objetivo, podemos considerar tomar x como solución inicial e y como solución guía e iniciar el nuevo camino desde x hasta y. Para seleccionar los movimientos no se considera la función objetivo o el criterio utilizado hasta el momento, sino que se incorporan a x atributos de y, construyendo una

trayectoria de x a y, esperando que alguna de las soluciones intermedias que se visitan en este proceso de "entorno constructivo"sea muy buena.

Es importante destacar el hecho de que muchas de las aplicaciones basadas en *TS* no utilizan los últimos elementos descritos, por lo que son susceptibles de ser mejoradas. Al mismo tiempo, los éxitos de las numerosas implementaciones del procedimiento han llevado la investigación hacia formas de explorar con mayor intensidad sus ideas subyacentes.

2.5. Templado simulado

Kirpatrick, Gelat y Vecchi (1983) proponen un procedimiento o nueva estrategia heurística utilizada para la resolución de complejos problemas combinatorios, con el fin de obtener soluciones aproximadas a problemas de optimización. Este método es el denominado *Templado Simulado* (*"Simulated Anneling"(SA)*, en inglés), y surge como consecuencia del establecimiento de una analogía entre este tipo de problemas y resultados teóricos de la Termodinámica.

El *SA* ha sido probado con éxito en numerosos problemas de optimización, mostrando gran "habilidad"para evitar quedar atrapado en óptimos locales.

La idea original que dio lugar a esta metaheurística es el denominado *algoritmo de Metrópolis* [Metropolis et al., 1953]. Este algoritmo simula el cambio de energía en el proceso de enfriamiento de un sistema físico. Las leyes de termodinámica establecen que, a una temperatura T, la probabilidad de un aumento de energía de magnitud ∂E viene dada por la siguiente expresión.

$$P(\partial E) = \exp\left(\frac{-\partial E}{KT}\right), \tag{2.1}$$

donde K es la constante de Boltzmann.

La simulación de Metrópolis genera una permutación y calcula el cambio resultante en la energía. Si ésta decrece, el sistema se mueve al nuevo estado, en caso contrario, se acepta el nuevo estado de acuerdo con la probabilidad dada en la ecuación (2.1).

A principios de los años 80, Kirpatrick, et al. (1983), trabajando en el diseño de circuitos electrónicos y de modo independiente Cerny (1985), investigando el *TSP*, consideraron aplicar el algoritmo de Metrópolis en problemas de optimización combinatoria que aparecen en este tipo de diseños. Para ello establecieron una analogía entre los parámetros que intervienen en la simulación termodinámica, y los que aparecen en los métodos de optimización local, la cual se muestra en la Tabla 2.1.

Cuadro 2.1: Analogía entre la simulación termodinámica y los métodos de optimización

Termodinámica		Optimización
Configuración	$\Leftrightarrow$	Solución factible
Configuración fundamental	$\Leftrightarrow$	Solución óptima
Energía de la configuración	$\Leftrightarrow$	Coste de la solución

Para llevar a efecto esta analogía, establecen un paralelismo entre el proceso de las moléculas de una sustancia que van colocándose en los diferentes niveles energéticos buscando un equilibrio, y las soluciones visitadas por un procedimiento de búsqueda local.

Luego *SA* es básicamente una estrategia heurística de búsqueda local, en la cual la elección del nuevo elemento del entorno $N(x)$ se realiza aleatoriamente. Como este tipo de estrategia presenta el inconveniente de la posibilidad de caer en un óptimo local y no ser capaz de salir de él, el *SA* lo

evita permitiendo con una cierta probabilidad (cada vez menor conforme nos acercamos a la solución óptima) el paso a soluciones de no mejora. En efecto, analizando el comportamiento del factor de Boltzmann en función de la temperatura, como se muestra en la Figura 2.7, vemos que conforme disminuye ésta, disminuye rápidamente la probabilidad de que aceptemos una solución peor que la actual.

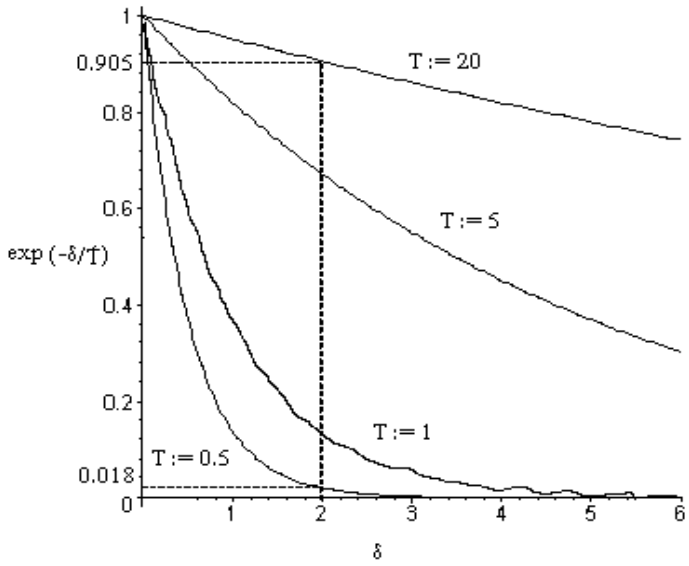

Figura 2.7: Valor del factor de Boltzmann en función de la temperatura T y de δ.

Se puede observar en la Figura 2.7 que para $\delta := 2$, es 50 veces menos probable un movimiento si $T := 0{,}5$ que si $T := 20{,}0$.

Por lo tanto, la estrategia que sigue el *SA* es partir de una "temperatura" alta, y posteriormente reducirla lentamente, disminuyendo la probabilidad de cambios a soluciones peores cuando se aproxima al óptimo buscado.

De esta forma *SA* tiene la habilidad de salir de óptimos locales al aceptar movimientos de no mejora en los estados intermedios. Al final del pro-

ceso estos son tan poco probables que no se producen, con lo que, si no hay movimientos de mejora el algoritmo finaliza. En la Figura 2.8 se muestra un esquema general del algoritmo para un problema de optimización [Martí, 2002].

```
Procedure Templado Simulado
    main {
        solución inicial x;
        temperatura inicial T;
        while (no congelado) {
            n := 0;
            while (n < L + 1) {
                                n := n + 1;
                                Sea x' ∈ N(x);
                                d := f(x') − f(x);
                                If d < 0
                                    x := x';
                                If d > 0
                                    x := x' con P := d^{−e/T};
            }
            Hacer T := rT;
    }
```

Figura 2.8: Algoritmo *SA*.

A continuación se determinan los parámetros del sistema.

. **Temperatura inicial**. Una de las características que debe cumplir toda heurística de búsqueda es la de no ser dependiente de la solución

inicial (T_0). Esto lo consigue *SA* partiendo de una temperatura inicial alta, con lo cual al principio irá erráticamente recorriendo soluciones lejanas de la óptima. Sin embargo, no parece conveniente considerar para T_0 valores fijos independientes del problema, se suele determinar realizando una serie de pruebas para alcanzar una determinada fracción de movimientos aceptados.

. **Velocidad de enfriamiento**. La velocidad de enfriamiento está representada por r.

. **Longitud** L. La longitud L se toma habitualmente proporcional al tamaño esperado de $N(x)$.

. **Temperatura final** T_f. Éste es el criterio para finalizar la secuencia de enfriamiento. En teoría, la temperatura correspondiente al "sistema frío"debería ser $T_f := 0$. Sin embargo, bastante antes de llegar a ese valor es prácticamente nula la probabilidad $\exp(-\delta/T)$ de que se acepte un movimiento hacia una solución peor. Por lo tanto, normalmente se puede realizar con $T_f > 0$, sin pérdida de calidad en la solución. Usualmente, se hace $cont := cont + 1$ cada vez que se completa una temperatura y el porcentaje de movimientos aceptados es menor de la cantidad *MinPercent*. Contrariamente, se hace $cont := 0$ cuando se mejora la mejor solución almacenada [Díaz et al., 1996][Martí, 2002].

La clave de una implementación de *SA* es el manejo de la cola o secuencia de enfriamiento. Cuál es la T_0 y cómo disminuye la temperatura en cada iteración, son las dos preguntas a responder para diseñar un algoritmo [Martí, 2002].

Las últimas implementaciones de *SA* apuntan a olvidarse de la analogía física y manejar la cola de enfriamiento como una estructura de memoria.

Es decir, la probabilidad de aceptar o rechazar un movimiento de no mejora depende no del tiempo transcurrido sino de lo sucedido en la búsqueda. En este sentido, la probabilidad será función de algunas variables de estado. En la actualidad se están diseñando numerosos algoritmos híbridos donde la búsqueda local se realiza con un procedimiento basado en *SA*, en ocasiones combinado con *TS*.

2.6. GRASP

Los métodos *GRASP* fueron desarrollados al final de la década de los años 80, con el fin de resolver problemas difíciles en el campo de la optimización combinatoria [Díaz et al., 1996].

El término *GRASP* fue introducido por Feo y Resende (1995), en un artículo en el que demuestran como desarrollar tal heurística para problemas de optimización combinatoria.

La palabra *GRASP* proviene de las siglas de *Greedy Randomized Adaptive Search Procedures*, una traducción de las palabras que forman el acrónico sería "Procedimientos de búsqueda, adaptativos, aleatorizados y voraces". Cada una de estas palabras caracteriza una de las componentes de esta metaheurística, la cual dirige la mayor parte de su esfuerzo a construir soluciones de alta calidad que son posteriormente procesadas con el fin de conseguir otras aún mejores. Es decir, cada iteración de *GRASP* consiste de dos fases, una fase de construcción y otra de post-procesamiento o mejora.

En la primera fase se construye iterativamente una solución posible, considerando un elemento en cada paso. En cada iteración la elección del próximo elemento para ser añadido a la solución parcial se determina a través de una función *greedy*. Esta función mide el beneficio de añadir cada uno de los

elementos según la función objetivo y elegir la mejor.

Se dice que un algoritmo es aleatorizado si su respuesta está determinada no sólo por los datos de entrada, sino también por los valores producidos por un generador de números aleatorios. En *GRASP* se dice que el heurístico es aleatorizado porque no selecciona el mejor candidato según la función *greedy* adaptada, sino que, con el objeto de diversificar y no repetir soluciones en dos construcciones diferentes, se construye una lista con los mejores candidatos de entre los que se toma uno al azar.

El término adaptativo ha sido usado en el contexto de sistemas de control, ya que en un número de campos muy importantes de la ingeniería es necesario construir controles para sistemas que muestran con el tiempo cambios en sus características dinámicas, cambios que se deben, en la mayoría de los casos, a un medio ambiente en continua evolución.

En este caso se dice que el heurístico se adapta porque en cada iteración se actualizan los beneficios obtenidos al añadir el elemento seleccionado a la solución parcial.

En la fase de mejora se suele emplear un procedimiento de intercambio simple con el objeto de no emplear mucho tiempo en esta mejora.

Una descripción de este procedimiento se muestra en la Figura 2.12.

2.7. Otros métodos

2.7.1. Relajaciones

Algunas herramientas para la búsqueda de soluciones de problemas de optimización aprovechan soluciones óptimas de problemas simplificados del original. Estas simplificaciones se conocen como relajaciones. Es por ello que se asume que las relajaciones deben ser necesariamente problemas fáciles de

```
Procedure GRASP
    main {
        Entrada;
        while (criterio de parada no satisfecho) {
            Construcción solución greedy aleatorizada (solución);
            Búsqueda local (solución);
            Actualizar la solución (solución, mejorsoluciónencontrada);
        }
        Return mejorsoluciónencontrada;
    }
```

Figura 2.9: Algoritmo genérico *GRASP*.

resolver.

Los métodos de relajación fueron introducidos por Agmon (1954) y Motzkin y Shoemberg (1954).

Cuando el problema relajado es un problema de optimización combinatoria, hablaremos entonces de una relajación combinatoria. En muchos casos la relajación es un problema que puede ser resuelto rápidamente.

La estructura del problema tratado en este trabajo, el *mAP*, es conveniente para la aplicación de una relajación como procedimiento para obtener cotas inferiores. Existen varios tipos de relajación, entre las cuales cabe destacar la relajación lineal y la relajación Lagrangiana.

Para resolver el problema planteado, haciendo uso de la relajación lineal, no se considera la condición de integrabilidad sobre el vector $\mathbf{x}$ y la forma apropiada de resolver este problema haciendo uso de la relajación Lagrangiana, sería dualizar uno o varios conjuntos de restricciones. La ventaja de estas relajaciones Lagrangianas, algunas veces llamadas *dual Lagrangiano*,

es que una buena aproximación a esta cota puede muchas veces ser calculada con mucho menos esfuerzo que el involucrado en la solución de la relajación lineal.

Tipos de relajaciones

Geoffrion (1974) definió una relajación de un problema de minimización de la siguiente forma [Guignard, 2003].

Definición 2.7.1 *Considérense los problemas de optimización siguientes:*

$$\text{mín}\,\{z(\boldsymbol{x}) : \boldsymbol{x} \in S\} \tag{2.2}$$

$$\text{mín}\,\{w(\boldsymbol{x}) : \boldsymbol{x} \in S'\}\,. \tag{2.3}$$

Se dice que (2.2) es una relajación de (2.1), si $S \subseteq S'$ *y* $w(\boldsymbol{x}) \leq z(\boldsymbol{x})$ *para todo* $\boldsymbol{x} \in S$.

Las relajaciones de un problema original de minimización pretenden proveer, de forma rápida, cotas inferiores del valor óptimo de éste; ya que:

$$\min_{\mathbf{x}\in S'} w(\mathbf{x}) \leq \min_{\mathbf{x}\in S} z(\mathbf{x}) \leq \min_{\mathbf{x}\in S} w(\mathbf{x})$$

Por ello se asume que las relajaciones deben ser problemas fáciles de resolver.

Una descripción del método de relajación es la siguiente: Considérese el sistema lineal $A\mathbf{x} \leq b$. Seleccionar un vector $\mathbf{x}^0$ y un escalar λ. Determínense vectores, $\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3$,..., inductivamente de la siguiente forma. Supón- gase $\mathbf{x}^i$ hallado ($i \geq 0$). Chequear cual $A\mathbf{x}^i \leq b$ es satisfecha. Si esto pasa se ha encontrado una solución y el método se detiene. Si no, se escoge una inecuación, $\mathbf{a}_k\mathbf{x} \leq \beta_k$ de $A\mathbf{x} \leq b$, violada por $\mathbf{x}^i$. Sea $\mathbf{x}^{i+1}$ dado por:

$$\mathbf{x}^{i+1} := \mathbf{x}^i \frac{\lambda(\beta_k - \mathbf{a}_k\mathbf{x}^i)}{||\mathbf{a}_k||}(\mathbf{a}_k)^t,$$

así, $\mathbf{x}^{i+1}$ se obtiene de $\mathbf{x}^i$ por la proyección del vector $\mathbf{x}^i$ sobre el hiperplano $\mathbf{a}_k\mathbf{x} \leq \beta_k$. Para $\lambda := 2$, $\mathbf{x}^{i+1}$ es la reflexión de $\mathbf{x}^i$ en este hiperplano. Si $\lambda := 1$, $\mathbf{x}^{i+1}$ es la proyección de $\mathbf{x}^i$ sobre este hiperplano.

Una gran utilidad de las cotas inferiores es que sirven para evaluar la bondad de las heurísticas, pero otra no menos importante es que sirven para acelerar algoritmos que buscan soluciones exactas del problema original.

Si P, Q y R representan problemas tales que R es una relajación de P y Q una relajación de R, entonces Q es una relajación de P, y en tal caso se dice que R es más ajustada a Q que P. En tal sentido, cuando se dispone de un problema de optimización difícil, suele convenir tener relajaciones lo más ajustadas posibles para dichos problemas.

Entre los tipos de relajaciones existentes, se encuentran:

. **Relajación lineal**: consiste en prescindir sólo de la restricción que requiere de la integrabilidad de las variables enteras del modelo.

. **Relajación por eliminación**: consiste en eliminar del modelo original una o varias restricciones, sin perder demasiado, de tal forma que el nuevo modelo resulte más sencillo de resolver.

. **Relajación subrogada**: consiste en sustituir una o varias restricciones mediante una combinación lineal de las mismas.

. **Relajación por descomposición**: consiste en dividir las restricciones en grupos y dividir los costes originales, mediante una combinación convexa entre los grupos, para luego resolver independientemente cada subproblema descrito por cada grupo.

. **Relajación Lagrangiana**: consiste en cambiar una o varias restricciones del modelo original mediante combinaciones lineales de las mismas. Sin embargo, a diferencia de la relajación subrogada, la nueva

combinación no se considera una restricción, sino su oportuna holgura como una penalización en la función objetivo para soluciones que satisfacen las restricciones eliminadas.

Proposición 2.7.2 *1. Si (2.2) es no factible, entonces (2.1) es no factible.*

2. Sea $\boldsymbol{x}^*$ una solución óptima de (2.2). Si $\boldsymbol{x}^* \in S$ y $w(\boldsymbol{x}^*) := z(\boldsymbol{x}^*)$, entonces $\boldsymbol{x}^*$ es solución óptima de (2.1).

Demostración.

1. Si (2.2) es no factible, entonces S':= $\emptyset$, pero como S$\subseteq$S':= $\emptyset$, se tiene que S:=$\emptyset$, así (2.1) es no factible.

2. Como $\mathbf{x}^* \in$ S, entonces $\text{mín}_{\mathbf{x}\in S}\, z(\mathbf{x}) \leq$ z($\mathbf{x}^*$):= w($\mathbf{x}^*$):=$\text{mín}_{\mathbf{x}\in S'}\, w(\mathbf{x})$, entonces $\text{mín}_{\mathbf{x}\in S}\, z(\mathbf{x}) \leq \text{mín}_{\mathbf{x}\in S'}\, w(\mathbf{x})$ pero como (2.2) es una relajación de (2.1), se tiene que $\text{mín}_{\mathbf{x}\in S'}\, w(\mathbf{x}) \leq \text{mín}_{\mathbf{x}\in S}\, z(\mathbf{x}) \therefore \text{mín}_{\mathbf{x}\in S'}\, w(\mathbf{x}) := \text{mín}_{\mathbf{x}\in S}\, z(\mathbf{x})$, así $\mathbf{x}^*$ es solución óptima de (2.1).

■

El papel de la relajación es doble: ella proporciona cotas sobre el valor óptimo de problemas difíciles, y sus soluciones, mientras normalmente la infactibilidad del problema original puede verse a menudo como los puntos de partida (guías) para el heurístico especializado (Guignard, 2003).

Relajaciones de programación lineal entera

Trataremos aquí problemas de programación lineal entera (PLE) o simplemente programación entera (PE), en los cuales el conjunto de restricciones es definido por un conjunto poliédrico racional más condiciones de integrabilidad sobre al menos un subconjunto de componentes de $\mathbf{x}$.

La relajación más ampliamente usada en un problema de PE es la relajación lineal, es decir, el problema entero ignorando la condición de integrabilidad sobre $\mathbf{x}$.

Definición 2.7.3 *Para el problema entero*

$$\text{mín}\left\{\boldsymbol{cx} : \boldsymbol{x} \in P \cap \mathbb{Z}^n\right\}, \tag{2.4}$$

con $P := \left\{\boldsymbol{x} \in \mathbb{R}^n_+ : A\boldsymbol{x} \geq \boldsymbol{b}\right\}$*, la relajación lineal es el problema lineal*

$$\text{mín}\left\{\boldsymbol{cx} : \boldsymbol{x} \in P\right\}. \tag{2.5}$$

Como P$\cap\mathbb{Z}^n \subseteq$ P y, la función objetivo de (2.4) y la de la relajación lineal es la misma, entonces por Definición 2.7.1, (2.5) es claramente una relajación de (2.4).

El problema relajado es obtenido por eliminación de las condiciones de integrabilidad de las variables. En muchos problemas las restricciones pueden ser particionadas en un conjunto de unas restricciones "simples"que pueden ser manipuladas "fácilmentez otras "complicadas". Una relajación puede ser obtenida removiendo las restricciones "complicadas.eincluyéndolas en la función objetivo, en tal sentido $\text{mín}_{\mathbf{x}\in P}\, w(\mathbf{x}) \leq \mathbf{cx}$, para todo $\mathbf{x} \in$ P$\cap\mathbb{Z}^n$.

Considérese el siguiente problema de programación entera (PE):

$$\begin{aligned} \text{mín}\, z &:= \mathbf{cx} \\ s.a : A\mathbf{x} &\geq \mathbf{b} \\ D\mathbf{x} &\geq \mathbf{d} \\ \mathbf{x} &\in \mathbb{Z}^n_+ \end{aligned} \tag{2.6}$$

Supóngase que las restricciones $A\mathbf{x} \geq \mathbf{b}$ de (2.6) son "buenas.en el sentido que un programa entero con exactamente estas restricciones es fácil. Así, si son eliminadas las "restricciones complicadas"$D\mathbf{x} \geq \mathbf{d}$ de (2.6), la relajación resultante es más fácil de resolver que el modelo (2.6). Entre los problemas que tienen tal estructura se pueden citar el problema del viajante de comercio

(TSP) (si son eliminadas las restricciones conectantes) y el problema de localización de plantas sin capacidades (si son eliminadas las restricciones de demandas de clientes), entre otros. Sin embargo, las cotas restantes obtenidas de la relajación pueden ser débiles, ya que algunas restricciones importantes son totalmente ignoradas. Un camino para enfrentar esta dificultad es la relajación lagrangiana.

Relajación Lagrangiana

Considérese el PE en una forma un poco más general

$$\begin{aligned} minz &:= \mathbf{cx} \\ \text{s.a:} D\mathbf{x} &\geq \mathbf{d} \\ \mathbf{x} &\in X, \end{aligned} \tag{2.7}$$

donde $D\mathbf{x} \geq \mathbf{d}$ son m restricciones complicadas y

$$X := \left\{\mathbf{x} : A\mathbf{x} \geq b; \mathbf{x} \in \mathbb{Z}_+^n\right\} \tag{2.8}$$

Para cualquier valor $\mathbf{u} := (u_1, ..., u_n)$ se define el problema $PE(\mathbf{u})$ por:

$$z(\mathbf{u}) := min_{\mathbf{x}\in X}(\mathbf{cx} + \mathbf{u}(\mathbf{d} - D\mathbf{x})) \tag{2.9}$$

Proposición 2.7.4 *El modelo (2.9) es una relajación de (2.7), para todo* $\boldsymbol{u} \geq 0$.

Demostración. El modelo (2.9) es una relajación del problema (2.7) si:

i) La región factible es al menos tan grande. Esto es cierto, ya que:

$$\{\mathbf{x} : D\mathbf{x} \geq \mathbf{d}, \mathbf{x} \in X\} \subseteq X$$

ii) El valor objetivo es a lo más tan pequeño en (2.9) como en (2.7), para toda solución factible de (2.7). Como $\mathbf{u} \geq 0$ y $D\mathbf{x} \geq \mathbf{d}$, para todo $\mathbf{x} \in X$, entonces $\mathbf{cx} + \mathbf{u}(\mathbf{d} - D\mathbf{x}) \leq \mathbf{cx}$, para todo $\mathbf{x} \in X \therefore z(\mathbf{u}) \leq z$.

■

En el modelo (2.9) las restricciones complicadas son manejadas por adición en la función objetivo como un término de penalización $\mathbf{u}(\mathbf{d} - D\mathbf{x})$, en otras palabras, $\mathbf{u}$ es el precio o variable dual asociado con las restricciones $D\mathbf{x} \geq \mathbf{d}$.

Definición 2.7.5 *Las variables duales* $\boldsymbol{u}$ *se denominan multiplicadores de Lagrange.*

Definición 2.7.6 *El problema (2.9) es llamado una relajación lagrangiana de (2.7) con parámetro* $\boldsymbol{u}$*.*

Intuitivamente, la relajación lagrangiana es el problema de hallar los multiplicadores de Lagrange $\mathbf{u}$, para compensar, penalizando, con $\mathbf{u}(\mathbf{d} - D\mathbf{x})$ el coste de soluciones $\mathbf{x}$ enteras, no negativas, que satisfacen $A\mathbf{x} \geq \mathbf{b}$, pero no satisfacen $D\mathbf{x} \geq \mathbf{d}$.

Como $z(\mathbf{u})$ es una cota inferior del valor óptimo de (2.7). Para hallar la mejor cota inferior sobre los posibles infinitos valores de $\mathbf{u}$, es necesario resolver el problema

$$w_{LD} := \text{máx}\left\{z(\mathbf{u}) : \mathbf{u} \geq 0\right\}, \tag{2.10}$$

el cual es llamado el *dual lagrangiano* de (2.7) con respecto a las restricciones $D\mathbf{x} \geq \mathbf{d}$.

Cuando las m restricciones que son dualizadas son igualdades de la forma $D\mathbf{x} := \mathbf{d}$, los correspondientes multiplicadores de Lagrange $\mathbf{u} \in \mathbb{R}^m$ son no

restringidos de signo, y (2.10) se transforma en:

$$w_{LD} := \max_{\mathbf{u}} z(\mathbf{u})$$

Proposición 2.7.7 *Si el vector* $\boldsymbol{u}$ *de los multiplicadores de Lagrange cumple para* $\boldsymbol{u} \geq 0$*,*

i.- $\boldsymbol{x(u)}$ *es una solución óptima de (2.9) y*

ii.- $D\boldsymbol{x(u)} \geq \boldsymbol{d}$ *y*

iii.- $(D\boldsymbol{x(u)})_i := \boldsymbol{d}_i$*, cuando* $\boldsymbol{u}_i > 0$ *(complementaridad),*

entonces $\boldsymbol{x(u)}$ *es óptima en (2.7).*

Demostración. Por (i) $w_{LD} \geq z(\mathbf{u}) := \mathbf{cx(u)} + \mathbf{u}(\mathbf{d} - D\mathbf{x(u)})$. Por (iii) $\mathbf{cx(u)}+\mathbf{u}(\mathbf{d}-D\mathbf{x(u)}) := \mathbf{cx(u)}$. Por (ii) $\mathbf{x(u)}$ es factible en (2.7) y así $\mathbf{cx(u)} \geq z$, luego $w_{LD} \geq \mathbf{cx(u)} + \mathbf{u}(\mathbf{d} - D\mathbf{x(u)}) := \mathbf{cx(u)} \geq z$, pero $w_{LD} \leq z \therefore z \leq w_{LD} \leq z \Rightarrow w_{LD} := z$ y así $\mathbf{x(u)}$ es solución óptima de (2.7). ■

De acuerdo con la Proposición 2.7.7, si las restricciones dualizadas son igualdades se satisface (iii) automáticamente, y así una solución óptima para (2.7) es óptima para (2.9) si ésta es factible en (2.9). Por lo tanto, en el *3AP*, como todas las restricciones son igualdades, la solución óptima de la relajación lagrangiana del *3AP* que sea factible para el *3AP* es óptima para el *3AP*.

Para entender el problema del dual lagrangiano (DL) se supone, por simplicidad, que X es un conjunto muy grande pero finito, por ejemplo $X := \{x_1, x_2, ..., x_T\}$.

Teorema 2.7.8 $w_{DL} := \min\{\boldsymbol{cx} : \quad D\boldsymbol{x} \geq \boldsymbol{d}, \quad \boldsymbol{x} \in conv(X)\}$.

Demostración. Por (2.10) $w_{DL} := \text{máx}_{\mathbf{u}\geq 0}\, z(\mathbf{u}) := \text{máx}_{\mathbf{u}\geq 0}(\text{mín}_{\mathbf{x}\in X}(\mathbf{cx} + \mathbf{u}(\mathbf{d} - D\mathbf{x}))) := \text{máx}_{\mathbf{u}\geq 0}(\text{mín}_{t:=1,...,T}(\mathbf{cx}_t + \mathbf{u}(\mathbf{d} - D\mathbf{x}_t))) := \text{máx}\,\eta$, donde $\eta \leq \mathbf{cx}_t + \mathbf{u}(\mathbf{d} - D\mathbf{x}_t)$, para todo $t := 1, ..., T, \mathbf{u} \in \mathbb{R}^T_+, \eta \in \mathbb{R}$. Escribiendo este problema como un problema de programación lineal (PL), se tiene:

$\text{máx}\,\eta$

sujeto a:

$$\begin{aligned} &\eta - \mathbf{u}(\mathbf{d} - D\mathbf{x}_t) \leq \mathbf{cx}_t, \qquad \text{para todo} \qquad t := 1, ..., T. \\ &\mathbf{u} \in \mathbb{R}^T_+, \qquad \eta \in \mathbb{R}. \end{aligned} \tag{2.11}$$

Tomando el dual de (2.11) queda:

$w_{DL} := \text{mín} \sum_{i:=1}^{T} \mu_t(\mathbf{cx}_t)$

sujeto a:

$$\begin{aligned} \sum_{t=1}^{T} \mu_t(D\mathbf{x}_t - \mathbf{d}) \quad &\geq 0 \\ \sum_{t=1}^{T} \mu_t \quad &:= 1 \\ \mu \in \mathbb{R}^T_+ & \end{aligned} \tag{2.12}$$

Haciendo $\mathbf{x} := \sum_{i:=1}^{T} \mu_t x_t$ con $\sum_{i:=1}^{T} \mu_t := 1, \mu \in \mathbb{R}^T_+$, se tiene:

$$\begin{aligned} w_{DL} &:= \text{mín}\,\mathbf{cx} \\ \text{s.a:} \quad D\mathbf{x} &\geq \mathbf{d} \\ \mathbf{x} &\in conv(X). \end{aligned}$$

■

Más generalmente, se puede mostrar que el resultado podría mantenerse cuando X es la región factible de cualquier *PE*.

$$X := \{\mathbf{x} \in \mathbb{Z}_+^n : A\mathbf{x} \geq \mathbf{b}\}$$

Este Teorema 2.7.8 dice cuan fuerte es una cota obtenida de dualización.

La demostración del Teorema 2.7.8, también dice una buena forma sobre la estructura del *DL*. Esto muestra que el problema tiene convexidad, así que se vuelve al *PL*. Además, se tiene que

$$w_{DL} := \text{máx}_{\mathbf{u}\geq 0}(\text{mín}_{t:=1,\ldots,T}(\mathbf{c}\mathbf{x}_t + \mathbf{u}(\mathbf{d} - D\mathbf{x}_t))).$$

Así el problema del *DL* puede verse como el problema de maximizar respecto a trozos de líneas convexas, pero $z(\mathbf{u})$ función no diferenciable.

La formulación lineal para el *DL* de la demostración del Teorema 2.7.8, produce una forma para calcular w_{DL}, aunque el gran número de restricciones significa que una restricción aproximada generada (o plano cortante) es requerida. Una aproximación alternativa que es muy simple y fácil de implementar, sin usar un sistema de *PL*, es un algoritmo de subgradiente.

Algoritmo del subgradiente

El algoritmo del subgradiente se diseña para resolver el problema de minimizar una función lineal convexa "en cuanto a partes".

$$\text{mín}_{\mathbf{u}\geq 0}(f(\mathbf{u})) \quad \text{donde} \quad f(\mathbf{u}) := \text{máx}_{t:=1,\ldots,T}[\mathbf{a}_t\mathbf{u} - b_t].$$

En el caso del *DL*, se tiene:

$$w_{DL} := \text{máx}_{\mathbf{u}\geq 0}\, z(\mathbf{u}),$$

donde $z(\mathbf{u}) := \text{mín}_{t:=1,\ldots,T}(\mathbf{c}\mathbf{x}_t + \mathbf{u}(\mathbf{d} - D\mathbf{x}_t)))$.

Un subgradiente es una generalización fácil de un gradiente.

Definición 2.7.9 *Un subgradiente en* $\boldsymbol{u}$ *de una función convexa* $f : \mathbb{R}^n \to \mathbb{R}$, *es un vector* $\gamma(\boldsymbol{u}) \in \mathbb{R}^n$ *tal que* $f(\boldsymbol{v}) \geq f(\boldsymbol{u}) + \gamma(\boldsymbol{u})(\boldsymbol{v} - \boldsymbol{u})^t$, *para todo* $\boldsymbol{v} \in \mathbb{R}^n$.

Para una función convexa continuamente diferenciable f,

$$\gamma(\mathbf{u}) := \nabla f(\mathbf{u}) := (\tfrac{\partial f}{\partial u_1}, \ldots, \tfrac{\partial f}{\partial u_m}),$$

es el gradiente de f en $\mathbf{u}$.

En la Figura 2.10 se describe el algoritmo del subgradiente para el problema dual lagrangiano.

Procedure SUBGRADIENTE

main {

Inicialización: $\mathbf{u} := \mathbf{u}^0$;

Iteración k : $\mathbf{u} := \mathbf{u}^k$;

Resolver el problema lagrangiano;

$PE(\mathbf{u}^k) : z(\mathbf{u}^k) := min_{\mathbf{x}\in X}(\mathbf{cx} + \mathbf{u}(\mathbf{d} - D\mathbf{x}))$;

con solución óptima $\mathbf{x}(\mathbf{u}^k)$;

$\mathbf{u}^{k+1} :=$ máx $\{\mathbf{u}^k + \mu_k(\mathbf{d} - D\mathbf{x}(\mathbf{u}^k)), 0\}$;

$k := k + 1$;

}

Figura 2.10: Algoritmo del subgradiente para el dual lagrangiano.

Se puede observar que el subgradiente de $z(\mathbf{u})$ en $\mathbf{u}^k$ es $\mathbf{d} - D\mathbf{x}(\mathbf{u}^k)$.

La simplicidad de este algoritmo es extraordinaria. En cada iteración se toma un paso del punto actual $\mathbf{u}^k$ en la dirección del subgradiente.

La dificultad está en la selección del tamaño de los pasos $\{\mu_k\}_{k:=1}^{\infty}$.

Para ello a continuación se muestran tres reglas para seleccionar μ_k:

a.- Si $\sum_k \mu_k \to \infty$ y $\mu_k \to 0$ cuando $k \to \infty$, entonces $Z(\mathbf{u}^k) \to w_{DL}$ (valor óptimo del DL).

b.- Si $\mu_k := \mu_0 \rho^k$ para algún parámetro $\rho < 1$, entonces $Z(\mathbf{u}^k) \to w_{DL}$ si μ_0 y ρ son suficientemente grandes.

c.- Si $\underline{w} \leq w_{DL}$ y $\mu_k := \frac{\epsilon_k[\underline{w} - z(\mathbf{u}^k]}{\|\mathbf{d} - D\mathbf{x}(\mathbf{u}^k)\|^2}$, con $0 < \epsilon_k < 2$, entonces $Z(\mathbf{u}^k) \to \underline{w}$, o el algoritmo encuentra $\mathbf{u}^k$ con $\underline{w} \leq Z(\mathbf{u}^k) \leq w_{DL}$ para algún k finito.

La regla (a) garantiza convergencia, pero como la serie $\{\mu_k\}$ puede ser divergente, convergencia es demasiado suave para ser de interés práctico real.

Por otro lado, los casos (b) y (c) llevan a una convergencia más rápida, pero cada vez con un posible inconveniente.

Usando el caso (b), el valor inicial de μ_o y ρ puede ser suficientemente grande, de otro modo, la serie geométrica $\mu_o \rho^k$ tiende a cero rápidamente, y la sucesión $\mathbf{u}^k$ converge antes de alcanzar un punto óptimo. En la práctica, antes que decrezca μ_k en cada iteración, un decrecimiento geométrico es realizado dividiendo por 2 el valor de μ_k cada ν iteraciones, donde ν es algún parámetro del problema natural tal como el número de variables.

Usando la regla (c), la dificultad es que la cota inferior dual $\underline{w} \leq w_{DL}$ es típicamente desconocida. Esto es más probable en la práctica que una buena cota superior primal $\overline{w} \geq w_{DL}$ sea conocida. Tal cota superior $\overline{w}$ es usada entonces inicialmente en lugar de $\underline{w}$. Sin embargo, si $\overline{w} > w_{DL}$, los términos $Z(\mathbf{u}_k) - \overline{w}$ en el numerador de la expresión para $\mathbf{u}_k$ podría no tender a cero, y así las sucesiones $\{\mathbf{u}_k\}$ y $\{Z(\mathbf{u}_k)\}$ podrían no converger. Si tal comportamiento es observado, el valor de $\overline{w}$ puede ser aumentado.

Como el algoritmo subgradiente frecuentemente finaliza antes que w_{DL} alcance el valor óptimo, y también como existe, en muchos casos, una función gap dual ($w_{DL} < z$) la relajación lagrangiana debe, típicamente, estar inmersa en un algoritmo de ramificar y acotar.

Una vez las variables duales $\mathbf{u}$ comienzan a aproximarse al conjunto de soluciones óptimas, una solución $\mathbf{x}(\mathbf{u})$ es obtenida finalizando con la existen-

cia de factibilidad primal cada vez que un subproblema lagrangiano $PE(\mathbf{u})$ es resuelto.

Construcción de una relajación Lagrangiana

En algunas ocasiones una reformulación previa a la relajación puede ayudar; y para muchos modelos complejos intuición y conocimiento de las interacciones entre las restricciones pueden proporcionar ingeniosos y eficientes esquemas de relajación.

Existen muchas formas para que un problema pueda ser relajado haciendo uso de una relajación Lagrangiana. Entre éstas tenemos (Guignard, 2003):

1.- Aislar un subproblema interesante y dualizar las otras restricciones. Este es el enfoque más usado, y tiene la ventaja que los subproblemas Lagrangianos son "interesantes", ya que tienen una estructura especial y pueden existir algoritmos especializados para resolverlos eficientemente.

2.- Si existen dos o más subproblemas interesantes y éstos tienen todas sus variables comunes, se pueden dividir las variables y entonces dualizar las restricciones copia. Primero se puede reformular el problema usando división de variables, renombrando las variables en parte de las restricciones como variables independientes.

La segunda forma de relajación, dada anteriormente, haciendo uso de la relajación Lagrangiana es llamada *descomposición Lagrangiana* (Soenen, 1977), *división de variables* (Näsberg et al., 1995) o *capas de variables* (Glover y Klingmann, 1988). Shepardson y Marsten (1980) y Ribeiro y Minoux (1996) están entre los primeros artículos que introducen este enfoque.

Cuando se dualizan restricciones igualdades, una solución Lagrangiana factible es automáticamente óptima para el problema entero original. Este es

el caso del 3AP donde todas las restricciones son igualdades.

2.7.2. Ramificar y acotar

En esta sección se darán los conceptos básicos de otra técnica algorítmica para resolver cualquier problema de programación entera, la cual divide y examina distintas zonas de la región factible, hasta encontrar una solución óptima.

Esta técnica, denominada *ramificar y acotar*, involucra una búsqueda sistemática bien estructurada de todo el espacio de soluciones factibles de problemas de optimización restringido que tienen un número finito de soluciones factibles [Gillett, 1976].

Ramificar y acotar fue introducida por Land y Doig (1960) y modificado por Dakin (1965). Es considerada como la primera herramienta para abordar problemas *NP*-difíciles, como el *3AP*, debido a que los problemas combinatorios tienen un número finito de soluciones y una técnica enumerativa, en teoría, es capaz de encontrar el óptimo mediante un proceso recursivo de división de la región factible.

Definición 2.7.10 *Un algoritmo que divide y examina distintas zonas de la región factible* $\boldsymbol{P}$ *de un problema de programación lineal, es llamado un algoritmo enumerativo.*

Supóngase que se desea resolver el siguiente problema de programación entera:

$$z := min\,\{\mathbf{cx} : A\mathbf{x} := \mathbf{b}, \mathbf{x} \geq 0 \;\text{ y }\; x_h \in \mathbb{Z} \;\text{ para }\; h \in \{1, \ldots, n\}\} \qquad (2.13)$$

Sea $\mathbf{x}^*$ la solución óptima de la relajación lineal de (2.13). Si $\mathbf{x}^*$ es un vector de componentes enteras, entonces $\mathbf{x}^*$ es la solución óptima de (2.13).

En caso contrario, existe una componente x_h^* no entera y la solución óptima buscada coincide con la solución óptima de alguno de los siguientes dos problemas enteros:

$$min\{\mathbf{cx} : \mathbf{x} \in P, x_h \leq [x*_h]\}$$

$$min\{\mathbf{cx} : \mathbf{x} \in P, x_h \geq [x*_h] + 1\}$$

Aplicando recíprocamente esta idea se obtiene la base del algoritmo conocido con *algoritmo de enumeración implícita*.

De lo anterior se puede concluir que no es necesario memorizar todos los puntos enteros obtenidos, sino que basta con almacenar el punto $\bar{\mathbf{x}}$ de menor coste en cada momento, el cual será finalmente la solución óptima de (2.13).

Un algoritmo de enumeración implícita se puede representar a través de un árbol, como se muestra en la Figura 2.11, el cual es llamado *árbol decisional*, donde cada nodo representa uno de los problemas de la ramificación

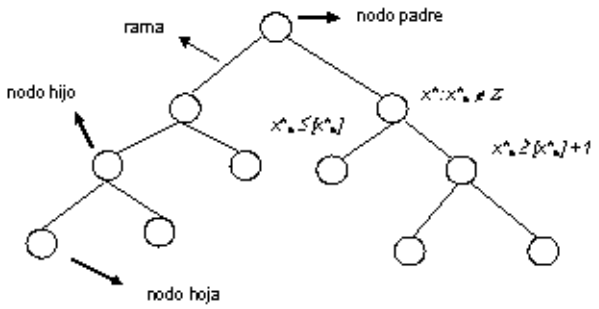

Figura 2.11: Árbol decisional

.

Definición 2.7.11 *Las líneas que unen dos nodos de un árbol decisional se denominan ramas.*

Definición 2.7.12 *El nodo que genera los subproblemas se llama nodo padre y los nodos que representan cada subproblema generado nodos hijos.*

Definición 2.7.13 *El nodo del árbol decisional que representa la relajación lineal de 2.13 se denomina nodo raíz y aquellos nodos que no tienen hijos, es decir, que no se ramifican, nodos hojas.*

Proposición 2.7.14 *Si* $P := \left\{\boldsymbol{x} \in \mathbb{R}^n_+ : A\boldsymbol{x} \leq \boldsymbol{b}\right\}$ *es acotado, un árbol decisional, como el mostrado en la Figura 2.11, será finito en cada nodo* i *que requiera división, una dicotomía de la forma* $(x_h \leq [x*_h], x_h \geq [x*_h]+1)$ *es seleccionada cuando* x_h *no es entera. En particular, si* $w_h := [max\{x_h : \boldsymbol{x} \in P\}]$, *ningún camino del árbol puede contener más de* $\sum_{h \in N} w_h$ *aristas.*

Demostración. Una vez agregada la restricción $x_h \leq d$ para algún $d \in \{0, \ldots, w_h - 1\}$, las únicas otras restricciones que pueden subsecuentemente aparecer en un camino de la raíz a una rama del árbol son $x_h \leq d'$ para $d' \in \{0, \ldots, d-1\}$ y $x_h \geq \bar{d}$ para $\bar{d} \in \{1, \ldots, d\}$. Sigue que el mayor número de restricciones que involucran x_h ocurrirá agregando $x_h \leq d$ para todo $d \in \{0, \ldots, w_h - 1\}$, o $x_h \geq d$ para todo $d \in \{1, \ldots, w_h\}$; o $x_h \geq d$ para todo $d \in \{1, \ldots, \alpha\}$ y $x_h \leq d$ para todo $d \in \{\alpha, \ldots, w_h - 1\}$. En cada uno de estos casos se requieren w_h restricciones en x_h y así se recorrerán $\sum_{h \in N} w_h$ aristas sobre cualquier camino. ■

Un subproblema deja de ramificarse, es decir, coincide con un nodo hoja, si se cumple alguna de las siguientes condiciones:

- Su solución óptima es entera.

- No contiene soluciones enteras con mejor valor, de la función objetivo, que alguna solución entera ya obtenida.
- Es no factible.

Ahora, si $\mathbf{c}$ es un vector de componentes enteras, entonces la segunda condición es equivalente a verificar la desigualdad $[\mathbf{cx}] + 1 \leq \mathbf{c}\bar{\mathbf{x}}$, donde $\mathbf{c}\bar{\mathbf{x}}$ es una cota superior del problema entero y es una restricción por exceso del valor óptimo de éste y $z^* := \mathbf{cx}^*$ es una cota inferior del nodo asociado al subproblema y representa una estimación por defecto del valor objetivo de cualquier solución entera de este subproblema.

Las cotas del valor óptimo de un problema de programación lineal entera obtenidas de la ramificación, permiten descartar del estudio algunos subproblemas que no contienen la solución óptima entera y localizar esta solución, este proceso es llamado *acotación* y constituye un ingrediente fundamental en cualquier proceso de enumeración implícita para evitar una enumeración explícita total.

Definición 2.7.15 *La unión del proceso de ramificación y el proceso de acotación definen el llamado algoritmo de ramificar y acotar.*

Algoritmo

Cuando la región factible P es un politopo, el algoritmo de ramificar y acotar es una técnica que precisa un número finito de pasos, y por tanto es un algoritmo. Sin embargo, el árbol decisional que representa el proceso de resolución del problema, puede contener un número exponencial de nodos hijos, lo que nos dice que no es una técnica polinomial. Por otro lado, la técnica ofrece ingredientes, como la acotación y la selección del problema a ramificar, capaces de ser ajustados apropiadamente en muchas situaciones

prácticas para minimizar el número de ramificaciones y por tanto producir una metodología efectiva en la resolución de problemas *NP*-difíciles como el *3AP*.

En la Figura 2.12 se describe en forma general el algoritmo de ramificar y acotar.

```
Procedure RAMIFICAR Y ACOTAR
Inicio {
    Inicializa Γ con el problema entero original y sea z̃ := ∞;
    while (Γ ≠ ∅) {
        extrae un subproblema de la lista Γ;
        calcula un óptimo x* de la relajación y sea z* := cx*;
        if (z* < z̃) {
            if (x* es entero) {
                z̃ := z* y x̃ := x*;
            }
            else {
                elige h tal que x*_h ∉ Z^n;
                añade a Γ los subprob con x_h ≤ [x*_h] y con x_h ≥ [x*_h] + 1;
            }
        }
    }
}
```

Figura 2.12: Algoritmo de ramificar y acotar.

Capítulo 3

Algoritmos genéticos para el *3AP*-axial

3.1. Introducción

El *3AP-axial*, como se dijo anteriormente, es un problema *NP-duro* con grandes áreas de aplicación y gran importancia en las operaciones prácticas. En el presente capítulo proponemos y desarrollamos nuevas herramientas para su resolución haciendo uso de los Métodos Evolutivos a través de dos algoritmos genéticos.

Para resolver un problema cualquier algoritmo genético debe tener los siguientes componentes básicos descritos en la Subsección 2.3.1:

. Una apropiada codificación.

. Valores de los parámetros.

. Una población inicial de poblaciones.

. Una función de evaluación.

. Operadores genéticos.

3.2. Metodología

En el presente trabajo se han desarrollado y analizado dos algoritmos genéticos, llamados genético I y genético II, haciendo uso de una metodología de desarrollo de software, que comprendió las siguientes fases: diseño, construcción/codificación y prueba [Fabregas, 1991] [Pressman, 2002].

La fase de diseño contempló la definición de variables y parámetros, y la estructura modular y de datos. En ésta se llevó a cabo la estructura de los algoritmos genéticos, los cuales fueron desarrollados a través de arreglos y estructuras, funciones y procedimientos.

La fase de codificación de los programas en sí, fue llevada a cabo de la siguiente manera: ambos algoritmos genéticos, denominados genético I y genético II, fueron codificados en C^{++}, ya que C^{++} es un lenguaje portable, lo cual hace que el programa se pueda ejecutar en un ordenador distinto a donde fue creado o generado. Otras razones por las cuales se escogió este lenguaje son: C^{++} trabaja como un lenguaje ensamblador haciendo más rápido los cálculos.

La fase de prueba consideró tres tipos:

a Prueba de unidad: validó el rendimiento funcional de los programas, ésta se realizó a través de los resultados arrojados por los programas, es decir, verificación de cada uno de los objetivos planteados.

b Prueba de integración: constituyó un medio para evaluar que las funciones y procedimientos fuesen ejecutados a cabalidad.

c Prueba de validación: comprobó que se cumplieran todos los requer-

imientos a través de comparaciones de los resultados con los obtenidos con algoritmos exactos y heurísticos de la literatura.

3.3. Algoritmos genéticos

Para el diseño e implementación de los algoritmos genéticos para resolver el *3AP-axial*, se llevó a cabo una investigación basada en los aportes teóricos existentes y la necesaria descripción, análisis e interpretación de la metaheurística algoritmos genéticos.

3.3.1. Representación o codificación

El alfabeto seleccionado para la representación genética del problema es el conjunto $N := \{1, 2, ..., n\}$, cada individuo es representado por una tripleta (i, j, k) (cromosoma), con $i, j, k \in N$, donde i, j, k son los genes del cromosoma y a cada tripleta se le asigna su coste c_{ijk}.

3.3.2. Parámetros de los algoritmos

Los siguientes parámetros definen al entorno evolutivo de los algoritmos:

n:	cardinalidad de N
mutación:	número de mutaciones a realizar
prob.:	probabilidad de selección
padres:	matriz de soluciones
fin:	número de generaciones a evaluar

3.3.3. Población inicial

La selección de la población inicial se realizó a través del método estocástico remanente con reemplazo, ya que un padre puede formar parte de otra solución o individuo.

Este método fue utilizado en ambos algoritmos. Sin embargo, la selección de la solución inicial es diferente en cada caso.

Solución inicial para el algoritmo genético I.

Los costes de cada individuo fueron generados aleatoriamente.

Para llevar a cabo este algoritmo se diseñó una matriz de soluciones (padres); es decir, ésta contiene todos los posibles cromosomas con mejor probabilidad que podrían formar parte de la solución. Esto para ir seleccionándolos como pertenecientes a la población inicial de individuos.

Con el fin de formar parte de la solución inicial, también se consideraron los cromosomas pertenecientes a la mejor solución obtenida a través de un algoritmo *greedy* adaptado al *3AP*, que en forma general se muestra en la Figura 3.1.

A continuación se presenta una descripción del procedimiento desarrollado en este algoritmo *greedy*.

Para cada $j = 0, 1, \ldots, n$, se elige un par (i,k) con el menor valor c_{ijk}, descartando los valores de i y k considerados con anterioridad. Este proceso se repite hasta completar la asignación. Luego se considera el segundo menor valor c_{ijk}, para cada $j = 0, 1, \ldots, n$, y se procede de igual forma. Al completar la asignación, se comparan los dos costes totales obtenidos tomándose la mejor solución como parte de la matriz de soluciones para la población inicial de individuos.

```
Procedure GREEDY 3AP
    Leer n, semilla;
    Generar costes aleatorios;
    Repetir
        Realizar las asignaciones;
        Evaluar las asignaciones;
        Seleccionar las asignaciones de menor coste;
        Eliminar los índices de la asignación seleccionada;
    Hasta completar la asignación
```

Figura 3.1: Algoritmo *greedy* adaptado al *3AP*.

Solución inicial para el algoritmo genético II.

En este algoritmo se trabajó con problemas existentes en la literatura (Balas y Saltzman, 1991; Burkard et al., 1996b; Crama y Spieksma, 1992), los cuales fueron generados uniformemente y de diferentes formas por los autores respectivos.

Al igual que en el genético I, en este algoritmo se diseñó una matriz de soluciones (padres) pero en ella fueron incluidas las soluciones generadas por seis algoritmos *greedy* adaptados al *3AP* y desarrollados para ello.

El código general de estos algoritmos *greedy* coincide con el presentado en la Figura 3.1, con la salvedad que en lugar de las dos primeras instrucciones, "*Leer n, semilla*" y "*Generar costes aleatorios*", se leen archivos de costes existentes en la literatura.

A continuación se presenta la descripción de estos algoritmos llamados *Greedy A*, *Greedy B*, *Greedy C*, *Greedy D*, *Greedy E* y *Greedy F*.

Para seleccionar los cromosomas a pertenecer a la solución, es decir, las tripletas que conformarán la solución de cada uno de estos algoritmos, se fija

una de las componentes y se varían las otras dos, cuyo orden de variación es permutado para obtener un nuevo algoritmo, seleccionándose para cada componente fija la tripleta de menor coste, hasta completar la asignación y se calcula el coste total.

La componente fija y el orden de las otras dos componentes en cada uno de estos algoritmos *greedy* se muestra en la Tabla 3.1.

Cuadro 3.1: Componentes fijas y orden de selección de las otras componentes en los algoritmos *Greedy A*, *Greedy B*, *Greedy C*, *Greedy D*, *Greedy E* y *Greedy F*.

	Algoritmos *Greedy*					
	A	B	C	D	E	F
Componente fija	i	i	j	j	k	k
Orden de variación	$j-k$	$k-j$	$i-k$	$k-i$	$j-i$	$i-j$

3.3.4. Operadores genéticos

Selección.

El operador de selección en los algoritmos genéticos, como su nombre lo indica, selecciona individuos para la reproducción. Esta selección está basada en la aptitud de los individuos; es decir, los individuos más aptos tienen mayor probabilidad de ser seleccionados para la reproducción.

En nuestros algoritmos la evaluación de un individuo (i,j,k) con $i, j, k \in N$, está dada por su coste c_{ijk} y la probabilidad de selección p_{ijk} se calcula a través de la siguiente ecuación:

$$p_{ijk} := \frac{c_{ijk}}{z} \tag{3.1}$$

Para el algoritmo genético I se procede de la siguiente forma: para seleccionar los primeros individuos para la reproducción, se parte de la solución obtenida por el algoritmo *greedy* y seleccionando otra solución del conjunto de soluciones factibles, se realizan comparaciones a través de los costos respectivos y de la probabilidad de selección de buenos individuos dada en la ecuación (3.1), seleccionándose para su reproducción aquellos cuya probabilidad sea menor que un valor uniformemente distribuido entre 0 y 1 generado aleatoriamente [Gonzalez y Centeno, 2001].

Luego de la reproducción de estos individuos seleccionados y de haber obtenido una nueva solución, a ésta se le aplica nuevamente este operador, seleccionándose así los nuevos individuos para la reproducción. Este proceso se repite hasta que se obtenga una "buena solución".

En el algoritmo genético II se procede de la siguiente manera: para seleccionar los primeros individuos para la reproducción, se consideran los individuos pertenecientes a cada una de las soluciones obtenidas por los seis algoritmos *greedy* utilizados en la solución inicial, se calcula la probabilidad de selección de cada uno de ellos haciendo uso de la ecuación (3.1) a través de la siguiente ecuación:

$$p_{ijk} := \frac{c_{ijk}}{\text{mín}\, z} \tag{3.2}$$

donde mín z representa el menor valor objetivo entre los mínimos obtenidos por cada algoritmo *greedy*, seleccionándose para su reproducción aquellos cuya probabilidad sea menor que un valor uniformemente distribuido entre 0 y 1 generado aleatoriamente [Gonzalez y Centeno, 2001].

En las siguientes generaciones de individuos se procede de igual forma que en el algoritmo genético I.

Mutación.

Para el algoritmo genético I el operador mutación implementado se describe a continuación:

Para la mutación se selecciona un individuo a mutar y se cambia el valor de los índices aleatoriamente, verificando que el nuevo cromosoma cumpla con las condiciones para formar parte de la solución, hasta completar la asignación, esto da como resultado un nuevo individuo al cual de inmediato le es evaluado su coste. Luego se compara con el individuo o solución anterior, realizándose este proceso hasta encontrar una solución con menor coste al de la solución anterior. Este algoritmo finaliza cuando no se puede mejorar más la solución o hasta que se realicen todas las mutaciones posibles.

Es de hacer notar, que en este caso, el número máximo de mutaciones introducido al algoritmo es de 5000 para n <10, 10000 para $10 \leq n$ <20, 20000 para $20 \leq n \leq$30 y 50000 para n >30.

Para el algoritmo genético II se procede de la siguiente forma:

A cada índice i, j, k, de un individuo a mutar, se le cambia su valor aleatoriamente, y de acuerdo a una cierta probabilidad, hasta completar la asignación, esto da como resultado un nuevo individuo al cual de inmediato le es evaluado su coste. Luego se compara con el individuo o solución anterior, realizándose este proceso hasta encontrar una solución con menor coste al de la solución anterior. Este algoritmo finaliza cuando no se puede mejorar más la solución o hasta que se realicen todas las iteraciones y/o mutaciones posibles.

3.3.5. Algoritmos

En esta subsección se presentan los algoritmos genéticos diseñados para resolver el *3AP*-axial haciendo uso de los componentes básicos descritos en las subsecciones desarrolladas anteriormente en esta sección.

El algoritmo genético I aplicado al *3AP* cuyos costes son generados aleatoriamente, y una de las soluciones que intervienen en la población inicial generada por el algoritmo *greedy* mostrado en la Figura 3.1, en forma general se presenta en la Figura 3.2.

```
Inicio
    Leer n, semilla;
    Leer número máximo de mutaciones;
    Generar costes aleatorios;
    Generar todas las posibles soluciones considerando las de mejor probabilidad;
    Generar solución a través del algoritmo greedy;
    Generar población inicial de individuos P(t);
    Repetir
        Seleccionar individuos a partir de P(t + 1);
        Realizar mutaciones;
        Evaluar individuo;
    Hasta última selección de individuos
Fin
```

Figura 3.2: Algoritmo genético I adaptado al *3AP*.

El algoritmo genético II aplicado al *3AP*-axial, el cual es aplicado a problemas existentes en la literatura y su población inicial generada por los seis algoritmos *greedy* descritos con anterioridad, se muestra en la Figura 3.3.

Inicio

Leer archivos de costes ;

Leer número máximo de mutaciones;

Generar solución a través de los seis algoritmos *Greedy*;

Generar población inicial de individuos $P(t)$;

Repetir

Seleccionar individuos a partir de $P(t+1)$;

Realizar mutaciones;

Evaluar individuo;

Hasta última selección de individuos

Fin

Figura 3.3: Algoritmo genético II adaptado al *3AP*.

3.4. Resultados computacionales

Con la finalidad de mostrar el desempeño de los algoritmos genéticos desarrollados en esta investigación, se realizaron comparaciones con algoritmos exactos y heurísticos existentes en la literatura.

En el genético I las pruebas se realizaron con 41 problemas generados aleatoriamente, y sus resultados comparados con los obtenidos con la aplicación de un algoritmo exacto codificado en LINGO, el cual es un lenguaje de modelos matemáticos en el que se pueden desarrollar, correr y modificar dichos modelos. Es de hacer notar, que los cuarenta y un (41) problemas fueron seleccionados previamente de cien (100) problemas generados aleatoriamente y que al ser resueltos a través de LINGO, se escogieron aquellos con los cuales se obtuvo solución óptima entera para $2 \leq n \leq 6$.

En la Tabla 3.2 se muestran los promedios de los resultados arrojados

por el genético I para $n := 5,6,10,14,16,20,25,30,40,50$, conjuntamente con los promedios de los resultados suministrados por el algoritmo exacto codificado en LINGO, para los 41 problemas generados aleatoriamente.

Cuadro 3.2: Resultados computacionales del algoritmo genético I.

n	*LINGO*		*Genético I*	
	valor	tiempo	valor	tiempo
5	**11974**	0.0000	**11974**	0.0028
6	**12724**	0.0000	**12724**	0.0030
10	—	—	24812	0.0033
14	—	—	25997	0.0050
16	—	—	28855	0.0069
20	—	—	27313	0.0094
25	—	—	27319	0.0300
30	—	—	30636	0.0430
40	—	—	25201	0.0970
50	—	—	32782	0.1188

Se observa en la Tabla 3.2 que el algoritmo genético I para $n := 5,6$ obtiene iguales resultados que el LINGO, es decir, obtiene la solución óptima, por lo cual podemos decir que a través del genético I se obtienen resultados satisfactorios debido a que se obtiene la solución óptima en tiempo de ejecución razonable. Por otro lado, se observa que el tiempo de ejecución en general es bastante pequeño, inferior a 1 segundo. El algoritmo en general funciona "bien"para un valor de n bastante elevado.

En el algoritmo genético II las pruebas se realizaron con problemas existentes en la literatura (Balas y Saltzman, 1991; Crama y Spieksma, 1992;

Burkard et al., 1996b) y se compararon los resultados con los presentados en Aiex et al. (2001) y Huang y Lim (2006).

En la Tabla 3.3 se muestran los resultados obtenidos por los seis algoritmos *Greedy* al ser ejecutados con los problemas de Balas y Saltzman (1991) (B-S), cuyos costes son generados uniformemente en el intervalo [1,100]. Es de hacer notar que se tomaron los promedios de los cinco problemas generados aleatoriamente, para cada n, por B-S.

Cuadro 3.3: Resultados computacionales de los algoritmos *Greedy* A, B, C, D, E, F con los problemas de B-S.

n	Algoritmos *Greedy*					
	A	B	C	D	E	F
12	105.0	103.2	132.4	101.8	**101.2**	103.2
14	85.0	**79.0**	115.0	83.2	112.2	96.4
16	103.2	83.8	92.2	77.2	73.2	**70.0**
18	105.6	107.0	**82.4**	92.6	108.8	86.8
20	**90.8**	92.4	110.0	96.8	92.6	96.8
22	104.8	83.2	89.4	101.6	100.8	**82.6**
24	115.4	93.2	**77.0**	110.6	100.0	131.8
26	88.2	123.6	85.2	103.2	106.4	**60.0**

En la Tabla 3.3 se observan resaltados los menores promedios de los valores obtenidos por los algoritmos *Greedy*, para los valores de $n := 12, 14, 16, 18, 20, 22, 24, 26$.

Otros problemas utilizados para la ejecución de estos seis algoritmos *greedy* y el algoritmo genético II, son los de la clase $T\triangle$, descritos por Crama y Spieksma (1992). Los costes de estos problemas son generados de la

siguiente forma: el *3AP* es visto como el problema de optimización sobre un grafo tripartito completo $K_{n,n,n}$. Para la clase $T\triangle$ una longitud $d_{uv} \geq 0$, es asignada a cada arista de $K_{n,n,n}$ y el coste c_{ijk} de un triángulo $(i, j, k) \in I\text{x}J\text{x}K$ es su longitud total, es decir, $c_{ijk} := d_{ij} + d_{jk} + d_{ki}$. Tres tipos de problemas generados aleatoriamente son considerados, ellos difieren en cómo las longitudes d_{uv} son computadas. Cada tipo consiste de tres ejemplos de tamaño $n := 33$ y tres ejemplos de tamaño $n := 66$, totalizando 18 problemas.

En la Tabla 3.4 se muestran los resultados obtenidos por los seis algoritmos *Greedy* al ser ejecutados con los problemas de Crama y Spieksma(C-S)-tipo I.

Cuadro 3.4: Resultados computacionales de los algoritmos *Greedy* A, B, C, D, E, F con los problemas de C-S-tipo I

Datos	n	Algoritmos *Greedy*					
		A	B	C	D	E	F
3DA99N1	33	2118	2118	**2079**	**2079**	2196	2196
3DA99N2	33	2284	2345	1740	1740	**1666**	**1666**
3DA99N3	33	2204	2204	2257	2257	**1993**	**1993**
3DA198N1	66	3301	**3297**	3622	3604	3632	3632
3DA198N2	66	3466	3466	3578	3578	**3244**	3360
3DA198N3	66	3958	3979	**3612**	**3612**	3958	3958

En la Tabla 3.4 se resaltan los menores valores obtenidos por los algoritmos *Greedy*, aplicados a los problemas generados aleatoriamente por C-S-tipo I, para los valores de $n := 33, 66$.

La Tabla 3.5 presenta los resultados obtenidos por los seis algoritmos *Greedy* al ser ejecutados con los problemas de C-S-tipo II.

Cuadro 3.5: Resultados computacionales de los algoritmos *Greedy* A, B, C, D, E, F con los problemas de C-S-tipo II

Datos	n	Algoritmos *Greedy*					
		A	B	C	D	E	F
3DIJ99N1	33	5246	5246	5204	5204	**5192**	**5192**
3DIJ99N2	33	5487	5487	**5430**	**5430**	5444	5444
3DIJ99N3	33	4794	4794	**4597**	**4597**	4752	4752
3DI198N1	66	10442	10434	10563	10560	**10330**	**10330**
3DI198N2	66	**9685**	**9685**	9729	9738	9689	9706
3DI198N3	66	10501	**10496**	10708	10706	10524	10561

En la Tabla 3.5 se observan resaltados los menores valores obtenidos por los algoritmos *Greedy*, aplicados a los problemas generados aleatoriamente por C-S-tipo II, para los valores de $n := 33, 66$.

En la Tabla 3.6 se muestran los resultados obtenidos por los seis algoritmos *Greedy*, al ser ejecutados con los problemas de Crama y Spieksma (C-S)-tipo III.

En la Tabla 3.6 se pueden ver resaltados los menores valores obtenidos por los algoritmos *Greedy*, aplicados a los problemas generados aleatoriamente por C-S-tipo III, para los valores de $n := 33, 66$.

El último bloque de problemas a los cuales se le aplicaron los algoritmos *greedy* A, B, C, D, E y F son los descritos por Burkard et al (1996). Estos problema tienen coeficientes de coste descomponibles. Sean α_i, β_j y γ_k los elementos de secuencias de n elementos de tres, entonces los coeficientes de coste son $c_{ijk} := \alpha_i.\beta_j.\gamma_k$. Cada coeficiente de coste entero tiene α_i, β_j y γ_k distribuidos uniformemente en el intervalo [0,10]. Para cada tamaño del

Cuadro 3.6: Resultados computacionales de los algoritmos *Greedy* A, B, C, D, E, F con los problemas de C-S-tipo III

Datos	n	Algoritmos *Greedy*					
		A	B	C	D	E	F
3D1299N1	33	151	149	148	150	**147**	149
3D1299N2	33	147	**144**	**144**	147	151	**144**
3D1299N3	33	**142**	145	149	147	144	147
3D1198N1	66	315	317	317	317	312	**309**
3D1198N2	66	316	315	312	308	314	**305**
3D1198N3	66	316	313	316	320	**312**	313

problema, todos los 100 problemas probados por Burkard, et al (1996) fueron considerados

En la Tabla 3.7 se presenta el promedio de los resultados obtenidos por los seis algoritmos *Greedy*, al ser ejecutados sobre los problemas de Burkard, Rudolf y Woeginger (B-R-W).

Cuadro 3.7: Resultados computacionales de los algoritmos *Greedy* A, B, C, D, E, F con los problemas de B-R-W.

n	Algoritmos *Greedy*					
	A	B	C	D	E	F
12	2465	2465	2529	2520	**2426**	2427
14	**2977**	**2977**	3040	3040	3046	3046
16	3181	3181	3204	3232	3118	**3117**

En la Tabla 3.7 se resaltan los menores volares obtenidos por los algorit-

mos *Greedy*, aplicados a los problemas generados aleatoriamente por B-R-W.

En la Tabla 3.8 se muestran los resultados obtenidos por el algoritmo genético II al ser ejecutado con los problemas de B-S, haciéndose la comparación con los resultados obtenido por B-S, el mejor resultado obtenido por Aiex, Resende, Pardalos y Toraldo (2001) (A-R-P-T) para 100 y 10000 iteraciones y los resultados producidos por los algoritmos de Huang y Lim (2006), el de búsqueda local multi-rondas, que se termina después de 100 rondas (multiLS) y el algoritmo genético híbrido (LSGA). Es de hacer notar, que se tomaron los promedios de los cinco problemas generados aleatoriamente, para cada n, por B-S.

Cuadro 3.8: Resultados computacionales del algoritmo genético II con los problemas de B-S, comparados con los resultados de B-S, A-R-P-T(100 iter.), A-R-P-T(10000 iter.), multiLS, LSGA y el Opt.

n	Opt	B-S	A-R-P-T (100 iter.)	A-R-P-T 10000(iter.)	multiLS	LSGA	genético II
12	15.6	24.0	18.0	15.6	26.2	15.6	35.0
14	10.0	22.4	15.0	10.0	26.4	10.0	11.4
16	10.0	25.0	18.4	10.2	26.0	10 0	14.8
18	6.4	17.6	17.2	7.2	24.6	7.2	12.8
20	4.8	27.4	18.6	6.4	26.8	5.2	20.4
22	4.0	18.8	20.8	7.2	26.4	5.6	20.8
24	1.8	14.0	16.8	7.4	23.2	3.2	15.8
26	1.3	15.7	21.2	8.4	23.2	3.6	17.0

En la Tabla 3.8 se observa que nuestro algoritmo mejora los resultados de (B-S) para n:= 14,16,18 y 20, de igual forma mejora los resultados de (A-

R-P-T), para 100 iteraciones, con n:= 14,16,18,24 y 26, obteniendo idéntico resultado que estos últimos para n:= 22 y también mejora los resultados de (multiLS) para n:= 14,16,18,20,22,24 y 26. Aunque en ningún caso mejora los resultados mostrados en (A-R-P-T), para 10000 iteraciones, y en (LSGA), se puede decir que nuestros resultados están bastante próximos a estos otros.

Por otro lado el promedio de mutaciones realizadas por el genético II fue de 4 mutaciones, aproximadamente, y el tiempo de cómputo inferior a 45s. Es de hacer notar, que aunque hemos desarrollado un algoritmo genético, los tiempos de cómputo registrados por nuestro algoritmo son pequeños.

En la Tabla 3.9 muestramos los resultados obtenidos por el algoritmo genético II al ser ejecutado con los problemas de C-S-tipo I, haciéndose la comparación con los resultados obtenido por éstos, una cota inferior (LB), el mejor resultado obtenido por A-R-P-T, para 100 y 10000 iteraciones, y los resultados producidos por los algoritmos de Huang y Lim (2006), multiLS y LSGA.

En la Tabla 3.9 se puede observar que nuestro algoritmo no mejora los resultados obtenidos por C-S, ni los obtenidos por A-R-P-T en ninguno de los dos casos y tampoco los alcanzados por los algoritmos multiLS y LSGA. Además, los tiempos de computo del genético II están entre 2 y 2:30 minutos para n:= 33 y de alrededor de 8 minutos para n:= 66. Por lo tanto, en este caso nuestro algoritmo no ha resultado eficiente.

En la Tabla 3.10 se muestran los resultados obtenidos por el algoritmo genético II, al ser ejecutado con los problemas de C-S-tipo II, haciéndose la comparación con los resultados obtenido por éstos, una LB, el mejor resultado obtenido por A-R-P-T para 100 y 10000 iteraciones y los resultados producidos por los algoritmos de Huang y Lim (2006), multiLS y LSGA.

En la Tabla 3.10 se observa que nuestro algoritmo genético no mejora

Cuadro 3.9: Resultados computacionales del algoritmo genético con los problemas de C-S- tipo I, haciéndose comparaciones con los resultados obtenido por éstos, una cota inferior (LB), el mejor resultado obtenido por A-R-P-T (100 y 10000 iter.) y los obtenidos por multiLS y LSGA .

Datos	n	LB	C-S	ARPT (100)	ARPT (10000)	multi LS	LSGA	gen. II
3DA99N1	33	1607	1618	1608	1608	1608	1608	2079
3DA99N2	33	1395	1411	1401	1401	1401	1401	1666
3DA99N3	33	1604	1609	1604	1604	1604	1604	1993
3DA198N1	66	2654	2668	2678	2664	2662	2662	3297
3DA198N2	66	2433	2469	2452	2449	2449	2449	3244
3DA198N3	66	2748	2775	2766	2758	2758	1758	3612

los resultados de C-S, ni los resultados obtenidos por A-R-P-T, en ninguno de los dos casos, así como tampoco los de multiLS y LSGA. Además, los tiempos de computo del genético II están entre 2 y 2:30 minutos para n:= 33 y superior a los 8 minutos para n:= 66 . Por lo tanto, en este caso nuestro algoritmo tampoco ha resultado eficiente.

En la Tabla 3.11 son mostrados los resultados obtenidos por el algoritmo genético II, al ser ejecutado con los problemas de C-S-tipo III, haciéndose la comparación con los resultados obtenido por éstos, una LB, el mejor resultado obtenido por A-R-P-T, para 100 y 10000 iteraciones, y los resultados producidos por los algoritmos de Huang y Lim (2006), multiLS y LSGA.

En la Tabla 3.11 se observa que nuestro algoritmo genético no mejora los resultados de C-S, ni los resultados obtenidos por A-R-P-T, en ninguno de los dos casos, así como tampoco los de multiLS y LSGA. Además, los

Cuadro 3.10: Resultados computacionales del algoritmo genético II con los problemas de C-S-tipo II y comparados con los resultados obtenido por éstos, una LB, el mejor resultado obtenido por A-R-P-T (100 y 10000 iter.) y los obtenidos por multiLS y LSGA.

Datos	n	LB	C-S	ARPT (100)	ARPT (10000)	multi LS	LSGA	gen.II
3DIJ99N1	33	4772	4861	4798	4797	4797	4797	5192
3DIJ99N2	33	5035	5142	5070	5067	5068	5067	5430
3DIJ99N3	33	4260	4352	4288	4287	4287	4287	4597
3DI198N1	66	9633	9780	9703	9694	9687	9684	10330
3DI198N2	66	8831	9142	8962	8951	8947	8944	9685
3DI198N3	66	9670	9888	9764	9751	9747	9745	10496

tiempos de computo del genético II están entre 2 y 2:30 minutos para n:= 33 y superior a los 9 minutos para n:= 66 . Por lo tanto, en este caso nuestro algoritmo no ha resultado eficiente.

En la Tabla 3.12 se muestran los resultados obtenidos por el algoritmo genético II, al ser ejecutado con los problemas de B-R-W, haciéndose la comparación con los resultados obtenido por éstos, el mejor resultado obtenido por A-R-P-T, para 100 y 10000 iteraciones, y los resultados producidos por los algoritmos de Huang y Lim (2006), multiLS y LSGA.

En la Tabla 3.12 se observa que el genético II no mejora los resultados de B-R-W, ni los de A-R-P-T, en ninguno de los dos casos, así como tampoco los de multiLS y LSGA. Sin embargo, tenemos que la desviación porcentual, comparando nuestros resultados con el mejor valor presentado, es de 7.3233 %, en promedio para los problemas tratados, lo que indica que

Cuadro 3.11: Resultados computacionales del algoritmo genético II con los problemas de C-S-tipo III y comparados con los resultados obtenido por éstos, una LB, el mejor resultado obtenido por A-R-P-T (100 y 10000 iter.) y los obtenidos por multiLS y LSGA.

Datos	n	LB	C-S	ARPT (100)	ARPT (10000)	multi LS	LSGA	gen. II
3D1299N1	33	133	135	133	133	133	133	147
3D1299N2	33	130	137	131	131	132	131	144
3D1299N3	33	130	135	131	131	131	131	142
3D1198N1	66	283	293	286	286	287	286	309
3D1198N2	66	281	294	286	286	286	286	305
3D1198N3	66	280	293	282	282	283	282	312

aunque no se han mejorado los resultados existentes, con el genético II se han obtenido resultados bastante cercanos al mejor existente. Además, los tiempos de cómputo son, en promedio, inferiores a 13 segundos, con lo cual podemos decir que con nuestro algoritmo hemos obtenido resultados bastante aceptables.

Cuadro 3.12: Resultados computacionales del algoritmo genético II con los problemas de B-R-W y comparados con los resultados obtenido por éstos, el mejor resultado obtenido por A-R-P-T (100 y 10000 iter.) y los obtenidos por multiLS y LSGA.

n	B-R-W	ARPT (100)	ARPT (10000)	multi LS	LSGA	gen. II
12	1188.02	1186.81	1186.81	1186.83	1186.81	1215.52
14	1469.19	1467.74	1467.74	1467.76	1467.74	1579.80
16	1476.80	1475.13	1475.13	1475.15	1475.13	1686.15

Capítulo 4

Algoritmos de búsqueda tabú para el *3AP*-axial

4.1. Introducción

Muchos problemas pueden formularse como un *3AP*, es por ello que se creó la necesidad de disponer de una herramienta que permitiera obtener, de manera rápida y eficiente, sino la óptima, una buena solución para el *3AP*; para ello se empleó la *Búsqueda Tabú* (*TS*), la cual es una metaheurística diseñada para obtener buenas soluciones a problemas de optimización combinatoria.

Así, en el presente capítulo proponemos y desarrollamos dos algoritmos metaheurísticos basados en *TS*, a fin de buscar "buenas"soluciones para el *3AP*-axial.

En muchos casos, la gran mayoría de los métodos descritos en el Capítulo 2 proporcionan soluciones muy próximas al óptimo y están entre los más eficaces. Esto ha hecho a *TS* sumamente popular entre aquellos interesados en buscar buenas soluciones a problemas combinatorios grandes que se presentan

en muchos casos particulares.

TS es una extensión de los métodos de búsqueda local. De hecho, el *TS* básico puede verse como una simple combinación de búsqueda local con memoria a corto plazo, teniendo como primeros elementos básicos la definición de su espacio de búsqueda y la estructura de la vecindad [Gendreau, 2003].

4.2. Metodología

La metodología empleada para realizar los algoritmos que aquí se presentan, *TS-I* y *TS-II* fue una combinación de la ingeniería del software, mediante el paradigma de ciclo de vida clásico de Pressman (1990) y ciclo de desarrollo de sistemas de información de Fábregas (1991). Esta comprendió tres fases: Diseño, Construcción/Codificación y Pruebas.

- Fase de Diseño: se diseñaron los procedimientos algorítmicos y las estructuras de datos a utilizar: arreglos tanto lineales como de dos y tres dimensiones. También se seleccionó el software de programación, los cuales fueron Borland C^{++}, para el primer algoritmo de búsqueda tabú (*TS-I*) y Microsoft Visual C^{++} para el segundo (*TS-II*).

- Fase de Construcción/Codificación: se implementaron los algoritmos diseñados en la fase anterior, en Borland C^{++} y Microsoft Visual C^{++}, respectivamente, haciendo uso de la programación modular. El componente de *TS* utilizado fue memoria a corto plazo y el tipo de memoria que se empleó fue explícita, es decir se conservaron las soluciones completas y consistió de todas las soluciones visitadas durante la búsqueda. El elemento de memoria a corto plazo empleado fue la estrategia para la lista de candidatos. Debido a que no se busca solucionar un problema determinado, se genera una matriz de costo y en cada ejecución

del programa se utilizan valores diferentes en dicha matriz. La solución inicial se obtiene a través de una función *greedy*, a partir de la cual se genera la vecindad. De ésta se selecciona una nueva solución, la cual es llamada solución actual, luego se obtiene la nueva vecindad donde es buscada la solución actual nueva y así sucesivamente hasta que se cumpla el criterio de parada. Finalmente, la mejor solución es aquella de menor coste de asignación entre todas las visitadas.

- Fase de Prueba: Se realizaron las pruebas necesarias para detectar errores que pudiesen presentarse tanto de lógica como de implementación, verificándose que los resultados producidos por una entrada definida fueran los requeridos. Además, se realizaron comparaciones de los resultados con los obtenidos por un algoritmo exacto codificado en LINGO y con resultados existentes en la literatura.

4.3. Algoritmos de búsqueda tabú (*TS*)

El desarrollo de los algoritmos TS para resolver el $3AP$-axial, que se presentan en este capítulo, se llevó a cabo basándose en los elementos básicos de TS como se presenta a continuación.

4.3.1. Espacio de búsqueda y estructura de la vecindad

La selección de un espacio de búsqueda y una estructura de vecindad, es el paso más crítico en el plan de cualquier heurístico TS, por tanto es en este paso donde uno debe hacer uso mejor de la comprensión y conocimiento del problema.

El *espacio de soluciones* de una búsqueda local o de una heurística TS es simplemente el espacio de todas las posibles soluciones que pueden ser

consideradas (visitadas) durante la búsqueda. Luego en el TS adaptado al $3AP$-axial, el espacio de búsqueda considerado es el conjunto de todas las soluciones factibles al problema, donde cada punto o elemento en el espacio de búsqueda corresponde a un conjunto de ternas que conforman una asignación. Por tanto, el número de elementos del espacio de búsqueda en los algoritmos TS aquí desarrollados es $(n!)^2$.

Estrechamente ligada a la definición del espacio de búsqueda está la de la *estructura de la vecindad*. A cada iteración de la heurística TS, las transformaciones locales que puedan aplicarse a la solución actual (S), definen un conjunto de soluciones vecinas ($N(S)$) en el espacio de búsqueda.

En nuestros algoritmos TS aplicados al $3AP$-axial, a partir de una solución actual se generan las candidatas (vecindad) que son soluciones "cercanas" a la solución actual, éstas se obtienen intercambiando los elementos de las ternas que conforman la solución uno a la vez. A medida que el espacio de soluciones aumenta su tamaño, la vecindad a utilizar es cada vez más pequeña con el fin de no revisar gran cantidad de soluciones.

En los algoritmos TS aquí desarrollados se tomó una vecindad de cinco elementos en las lista de candidatos, para $n < 10$, mientras que para $n \geq 10$ esta lista está conformada por una vecindad de tres elementos.

4.3.2. Estatus tabú

El *Estatus Tabú* es el elemento distintivo de TS cuando se compara con búsqueda local. Como se mencionó en la Sección 2.6, el estatus tabú es usado para prevenir el ciclado al moverse fuera del óptimo local a través de un movimiento de no-mejora. El hecho importante aquí es que cuando esta situación ocurre, se necesita hacer algo para impedir a la búsqueda rastrear sus pasos anteriores que invierta el efecto de recientes movimientos.

En los algoritmos *TS* aquí presentados, se utilizó memoria a corto plazo para tener en cuenta los atributos de soluciones que hayan sido cambiados en el pasado reciente, junto con una función que verifica si la solución actual pertenece a la historia de movimientos que evita revisitar soluciones. Los atributos seleccionados que se presentaran en soluciones recientemente visitadas se designan *tabú activos* y las soluciones que tengan ese tipo de elementos se convierten en *tabú*

4.3.3. Criterio de aspiración

El estatus tabú es un elemento importante en un algoritmo *TS*, pero a su vez demasiado poderoso, ya que éste puede prohibir movimientos atractivos, incluso cuando no hay peligro de que se produzca ciclaje, o pueden llevar a un estancamiento global del proceso de búsqueda. Es por ello que en estos algoritmos existe la necesidad de usar dispositivos algoritmicos que permitan la cancelación del estatus tabú. Estos dispositivos son llamados *criterio de aspiración.*

El criterio más simple de aspiración y el más frecuentemente utilizado consiste en permitir un movimiento, aún cuando tenga el status tabú, si éste produce una solución con un mejor valor objetivo que el de la mejor solución conocida actualmente, partiendo de que la nueva solución no se ha visitado previamente. Este es el criterio de aspiración utilizado en el desarrollo de los dos algoritmos *TS* presentados en este capítulo.

4.3.4. Criterio de parada

En teoría, la búsqueda de una mejor solución a través de una metaheurística *TS* podría seguir "para siempre", a menos que el valor óptimo del

problema sea conocido de antemano. En la práctica, obviamente, la búsqueda tiene que ser detenida en algún punto.

En nuestros algoritmos TS el *criterio de parada* implementado consiste en realizar la búsqueda, a partir de una solución inicial, haciendo uso de los elementos anteriores hasta regresar a dicha solución inicial.

4.3.5. Lista de candidatos

Un camino para controlar el número de movimientos examinados es por medio de la estrategia de *lista de candidatos*, que proporciona caminos estratégicos para generar un subconjunto $N'(S) \subseteq N(S)$ el cual se considera útil en la búsqueda de soluciones. El fracaso para dirigir los problemas involucrados creando listas de candidatos eficaces adecuadamente, es una de las más inminentes limitaciones que diferencia una aplicación de TS ingenua de una más sólidamente conectada con la realidad.

En los algoritmos metaheurísticos TS aquí presentados, se hace uso de una estrategia para la lista de candidatos que consiste en restringir el número de soluciones examinadas, disminuyendo la vecindad para los casos en los que el entorno de la solución es grande.

Como se mencionó en la Subsección 4.3.1, para $n < 10$ se consideró una vecindad de cinco elementos en la lista de candidatos, y para $n \geq 10$ esta lista está conformada por una vecindad de tres elementos.

4.3.6. Soluciones iniciales

Usualmente para generar una solución inicial es utilizada una heurística constructiva. Durante el proceso de construcción de la solución inicial, la longitud de las soluciones candidatas crece mientras sus costes de asignación decrecen. El procedimiento para cuando la solución obtenida es factible.

En este trabajo es utilizada una solución inicial generada a través de algoritmos *greedy*, para cada uno de los algoritmos *TS* desarrollados. Sin embargo, la selección de la solución inicial es diferente en cada caso.

Solución inicial para el *TS-I*. Los costes de cada tripleta fueron generados aleatoriamente y para seleccionar la solución inicial se consideraron las tripletas pertenecientes a la mejor solución obtenida a través del algoritmo *greedy* utilizado en el algoritmo genético I, que está descrito en la Subsección 3.3.3.

Solución inicial para el *TS-II*. La ejecución de este algoritmo se llevó a cabo considerando los mismos problemas de la literatura utilizados en el genético II, y se tomó como solución inicial la mejor solución entre las generadas por los algoritmos *Greedy A, Greedy B, Greedy C, Greedy D, Greedy E* y *Greedy F*, presentados en la Subsección 3.3.3, en cada uno de los casos.

4.3.7. Estrategia de intensificación

Dos componentes altamente importantes de largo plazo de *TS* son las estrategias de intensificación y de diversificación. Las estrategias de intensificación, las cuales son las que nos interesan en este trabajo, están basadas en la modificación de reglas de elección de tal forma que se favorezcan combinaciones de movimientos y características de solución que históricamente hayan sido buenas. Pueden además iniciar un regreso hacia regiones atractivas para buscar en ellas más extensamente.

En el algoritmo *TS-I* no es implementada este tipo de estrategia a largo plazo. Sin embargo, se han obtenido "buenas"soluciones del *3AP*-axial haciendo uso de la memoria a corto plazo y sus componentes descritos con anterioridad.

En el algoritmo *TS-II* aplicado al *3AP*-axial, se hace uso de la estrategia

de intensificación con el fin de, como su nombre la indica, intensificar la búsqueda de soluciones en regiones "atractivas".

El procedimiento de la estrategia de intensificación aplicado al *3AP*-axial se muestra en la Figura 4.1.

```
Procedure INTENSIFICACIÓN
        Seleccionar 10 mejores soluciones del algoritmo a corto plazo;
        Crear lista Soluciones Élite de mayor a menor;
        Guardar memoria a corto plazo de Soluciones Élite;
    Repetir
        Escoger mejor Solución Élite y quitarla de la lista;
        Aplicar TS a corto plazo, haciendo tabú movimiento previo a la solución;
        Si nueva solución válida para estrategia selección
            Si hay cupo agregar nueva solución a la lista;
            Si no reemplazar una solución de coste superior;
        Si nueva solución no válida para estrategia selección
            no se considera;
    Hasta lista vacía o criterio de parada
Fin
```

Figura 4.1: Estrategia de intensificación.

Descripción del procedimiento: luego de aplicar el algoritmo TS a corto plazo, son seleccionadas las 10 mejores soluciones obtenidas por este algoritmo, creando una lista de soluciones élite ordenada de mayor a menor y a su vez guardar la memoria a corto plazo que acompañó a cada una de las soluciones élite. Así, en este proceso de selección, la mejor solución encontrada por TS a corto plazo estará ubicada en la última posición de la lista.

Al escoger un elemento de la lista, se toma el último, es decir, la mejor solución encontrada hasta el momento y ésta es eliminada de la lista.

Como la memoria a corto plazo de esta solución ha sido guardada, se conoce la solución anterior que dió origen a ésta, por tanto ese movimiento se hace tabú, produciéndose así un nuevo camino aplicando *TS* a corto plazo.

Se compara la nueva solución con los elementos de la lista de soluciones élite, si su coste es menor que alguno de los existentes en la lista, se agrega si hay cupo y si no hay cupo se reemplaza en el lugar de uno con coste superior. Ahora, si la solución encontrada no mejora ninguna de las soluciones élite, no se considera, y nuevamente se verifica si la lista está vacía o se cumple el criterio de parada.

4.3.8. Algoritmos

En esta subsección se presentan los algoritmos *TS* diseñados para resolver el *3AP*-axial haciendo uso de los elementos básicos descritos en las subsecciones desarrolladas anteriormente en esta sección.

El algoritmo *TS-I* aplicado al *3AP*, cuyos costes son generados aleatoriamente y su solución inicial generada por el algoritmo *greedy* mostrado en la Figura 3.1, en forma general, se presenta en la Figura 4.2.

El algoritmo *TS-II* aplicado al *3AP*-axial, el cual es ejecutado sobre problemas existentes en la literatura y su solución inicial es la mejor entre las soluciones generadas por los seis algoritmos *greedy* descritos en la Subsección 3.3.3, se muestra en la Figura 4.3.

Inicio

Leer n, semilla;

Leer tamaño de vecindad;

Generar costes aleatorios;

Generar solución inicial S_0;

$S := S_0$;

Repetir

Generar vecindad de la solución actual *N(S)*;

Seleccionar $x' \in N(S)/f(x') < f(x), \forall x \in N(S) \nsubseteq HM$;

Actualizar Historia de los Movimientos (HM);

Hasta criterio de parada

Fin

Figura 4.2: Algoritmo *TS-I* adaptado al *3AP*.

4.4. Resultados computacionales

Al igual que en el caso de los algoritmos genéticos desarrollados en el Capítulo 3, el desempeño del *TS-I* y del *TS-II* se muestra a través de comparaciones con resultados obtenidos con algoritmos exactos y con los obtenidos a través de otras heurísticas y/o metaheurísticas.

Para el caso del *TS-I*, las comparaciones han sido realizadas con los resultados obtenidos haciendo uso de un algoritmo exacto codificado en LINGO y con los resultados del genético I, mientras que las comparaciones para el caso del *TS-II* se realizan con resultados de heurísticas existentes en la literatura y los obtenidos a través del genético II.

En el *TS-I* las pruebas se realizan con los 41 problemas generados aleatoriamente utilizados para la prueba del genético I.

Inicio

Leer archivos de coste;

Leer tamaño de vecindad;

Generar solución inicial S_0;

$S := S_0$;

Repetir

Generar vecindad de la solución actual *N(S)*;

Seleccionar $x' \in N(S)/f(x') < f(x), \forall x \in N(S) \nsubseteq HM$;

Actualizar Historia de los Movimientos (HM);

Hasta criterio de parada

Estrategia de intensificación Fin

Figura 4.3: Algoritmo *TS-II* adaptado al *3AP*.

En la Tabla 4.1 se muestran los promedios de los valores de los 41 problemas, obtenidos a través de LINGO, el genético I y el *TS-I*, para $n := 5, 6, 10, 12, 14, 16, 20, 22, 24, 26$.

En la Tabla 4.1 se observa que para $n := 5, 6$ se obtuvo la solución óptima con las metaheurísticas, también se puede observar que los valores obtenidos por el *TS-I* son menores y por tanto mejores que los obtenidos por el genético I, con lo cual podemos concluir que el *TS-I* genera mejores resultados que el genético I. Con relación a los tiempos de cómputo se observa que hasta $n := 30$ los tiempos registrados por el *TS-I* son menores que los registrados por el genético I, aunque luego cuando aumenta el tamaño del problema, el tiempo de cómputo tiende a ser mayor con el *TS-I* que con el genético I, pero siempre inferior en ambos casos a 1 seg., por lo tanto podemos decir que el algoritmo *TS-I* funciona eficientemente para valores grandes de n.

En el *TS-II*, al igual que en el genético II, las pruebas y comparaciones se

Cuadro 4.1: Resultados computacionales del algoritmo *TS-I*.

n	*LINGO*		*Genético I*		*TS-I*	
	valor	tiempo	valor	tiempo	valor	tiempo
5	**11974**	**0.0000**	**11974**	0.0028	**11974**	0.0001
6	**12724**	**0.0000**	**12724**	0.0030	**12724**	0.0017
10	—	—	24812	0.0033	**22906**	**0.0030**
14	—	—	25997	0.0050	**22272**	**0.0031**
16	—	—	28855	0.0069	**24370**	**0.0037**
20	—	—	27313	0.0094	**21052**	**0.0041**
25	—	—	27319	0.0300	**23071**	**0.0193**
30	—	—	30636	0.0430	**24810**	**0.0370**
40	—	—	25201	**0.0970**	**20603**	0.4951
50	—	—	32782	**0.1188**	**24916**	0.6256

realizaron con los problemas y resultados, respectivamente, de los artículos de Balas y Saltzman (1991), Crama y Spieksma (1992), Burkard et. al. (1996), Aiex et. al. (2001) y Huang y Lim (2006).

Como se expresó en la Subsección 3.3.3, la solución inicial para la implementación del *TS-II* se obtuvo seleccionando la mejor solución entre las obtenidas por los algoritmos *Greedy A, Greedy B, Greedy C, Greedy D, Greedy E* y *Greedy F*, presentados con anterioridad, para cada uno de los problemas aleatorios existentes en los artículos mencionados anteriormente y descritos en la Sección 3.4.

Los mejores resultados de estos algoritmos *greedy* para cada uno de los problemas, aparecen resaltados en la Tablas 3.3, 3.4, 3.5, 3.6 y 3.7.

En la Tabla 4.2 se muestran los resultados obtenidos por el algoritmo

TS-II al ser ejecutado con los problemas de B-S (Balas y Saltzman, 1991), haciéndose la comparación con los resultados obtenidos por B-S, el mejor resultado obtenido por A-R-P-T (Aiex et al., 2001) y de Huang y Lim (2006). Es de hacer notar que se tomaron los promedios de los cinco problemas generados aleatoriamente por B-S para cada n.

Cuadro 4.2: Resultados computacionales del algoritmo *TS-II* con los problemas de B-S, comparados con los resultados de B-S, A-R-P-T, el Opt, multiLS y LSGA.

n	Opt	B-S	A-R-P-T 100 iter.	A-R-P-T 10000 iter.	multiLS	LSGA	*TS-II*
12	15.6	24.0	18.0	15.6	26.2	15.6	38.8
14	10.0	22.4	15.0	10.0	26.4	10.0	35.2
16	10.0	25.0	18.4	10.2	26.0	10.0	30.8
18	6.4	17.6	17.2	7.2	24.6	7.2	33.2
20	4.8	27.4	18.6	6.4	26.8	5.2	34.2
22	4.0	18.8	20.8	7.2	26.4	5.6	30.6
24	1.8	14.0	16.8	7.4	23.2	3.2	23.8
26	1.3	15.7	21.2	8.4	23.2	3.6	23.6

En la Tabla 4.2 se observa que nuestro algoritmo no mejora los resultados de (B-S), de igual forma no mejora los resultados de (A-R-P-T) y tampoco mejora los resultados de (multiLS) y (LSGA). Sim embargo, en algunos casos nuestros resultados son muy próximos a algunos resultados de los otros autores.

En la Tabla 4.3 mostramos los resultados obtenidos por el algoritmo *TS-II* al ser ejecutado con los problemas de C-S-tipo I, haciéndose la comparación

con los resultados obtenidos por éstos, una cota inferior (LB), el mejor resultado obtenido por A-R-P-T y los resultados de los algoritmos multiLS y LSGA.

Cuadro 4.3: Resultados computacionales del algoritmo *TS-II* con los problemas de C-S-tipo I, haciéndose comparaciones con los resultados obtenido por éstos, una cota inferior (LB), el mejor resultado obtenido por A-R-P-T, multiLS y LSGA.

Datos	n	LB	C-S	ARPT 100	ARPT 10000	multi LS	LSGA	*TS-II*
3DA99N1	33	1607	1618	1608	1608	1608	1608	1815
3DA99N2	33	1395	1411	1401	1401	1401	1401	1474
3DA99N3	33	1604	1609	1604	1604	1604	1604	1685
3DA198N1	66	2654	2668	2678	2664	2662	2662	3032
3DA198N2	66	2433	2469	2452	2449	2449	2449	2947
3DA198N3	66	2748	2775	2766	2758	2758	2758	2947

En la Tabla 4.3 se observa que nuestro algoritmo no mejora ninguno de los resultados obtenidos por los otros algoritmos, cuando éstos se ejecutan sobre los problemas generados aleatoriamente por C-S-tipo I.

En la Tabla 4.4 mostramos los resultados obtenidos por el algoritmo *TS-II* al ser ejecutado con los problemas de C-S-tipo II, haciéndose la comparación con los resultados obtenido por éstos, una cota inferior (LB), el mejor resultado obtenido por A-R-P-T y los resultados de multiLS y LSGA.

En la Tabla 4.4 observamos que los resultados obtenidos por nuestro algoritmo *TS-II* mejora los de C-S. Asímismo, al comparar con los otros algoritmos, se puede observar que nuestros resultados están muy próximos a

Cuadro 4.4: Resultados computacionales del algoritmo *TS-II* con los problemas de C-S-tipo II y comparados con los resultados obtenido por éstos, una LB, el mejor resultado obtenido por A-R-P-T, multiLS y LSGA

Datos	n	LB	C-S	ARPT 100	ARPT 10000	multi LS	LSGA	*TS-II*
3DIJ99N1	33	4772	4861	4798	4797	4797	4797	4830
3DIJ99N2	33	5035	5142	5070	5067	5068	5067	5107
3DIJ99N3	33	4260	4352	4288	4287	4287	4287	4314
3DI198N1	66	9633	9780	9703	9694	9687	9684	9775
3DI198N2	66	8831	9142	8962	8951	8947	8944	9090
3DI198N3	66	9670	9888	9764	9751	9747	9745	10059

los obtenidos por éstos, ya que tienen, en promedio, un *Gap* igual a 0,0137, aplicados a los problemas generados aleatoriamente por C-S-tipo II.

En la Tabla 4.5 se muestran los resultados obtenidos por el algoritmo *TS-II*, al ser ejecutado con los problemas de C-S-tipo III, haciéndose la comparación con los resultados obtenido por éstos, una LB, el mejor resultado obtenido por A-R-P-T y los resultados de los algoritmos multiLS y LSGA.

En la Tabla 4.5 se puede observar que nuestro algoritmo *TS-II* mejora los resultados de C-S para todos los problemas de tamaño n:= 33 y para uno de los problemas con n:= 66. De igual forma, podemos ver que mejora, en un caso, el resultado de multiLS para n:= 33 y para los otros dos (2) problemas iguala el resultado. Para el problema 3D1299N1, nuestro algoritmo obtiene el mismo resultado que A-R-P-T, multiLS y LSGA, el cual coincide con la cota inferior reportada. De igual forma, se observa que para los otros dos (2) problemas con n:= 33, nuestra búsqueda tabú obtiene iguales resultados

Cuadro 4.5: Resultados computacionales del algoritmo *TS-II* con los problemas de C-S-tipo III y comparados con los resultados obtenido por éstos, una LB, el mejor resultado obtenido por A-R-P-T, multiLS y LSGA

Datos	n	LB	C-S	ARPT 100	ARPT 10000	multi LS	LSGA	*TS-II*
3D1299N1	33	133	135	133	133	133	133	133
3D1299N2	33	130	137	131	131	132	131	131
3D1299N3	33	130	135	131	131	131	131	131
3D1198N1	66	283	293	286	286	287	286	296
3DI1198N2	66	281	294	286	286	286	286	292
3D1198N3	66	280	293	282	282	283	282	300

que A-R-P-T y LSGA. Ahora, para $n{:=}\ 66$ el *TS-II* no mejora los resultados dados por A-R-P-T, multiLS, ni LSGA, sin embargo están muy próximos a estos valores. Todo esto al resolver los problemas generados aleatoriamente por C-S-tipo III.

En la Tabla 4.6 se muestran los resultados obtenidos por el algoritmo *TS-II*, al ser ejecutado con los problemas de B-R-W, haciéndose la comparación con los resultados obtenido por éstos, el mejor resultado obtenido por A-R-P-T y los reportados por multiLS y LSGA.

En la Tabla 4.6 se observa que nuestro algoritmo *TS-II* no mejora los resultados de los otros algoritmos al resolver los problemas generados aleatoriamente por B-R-W. Sin embargo, los valores obtenidos están muy próximos a los reportados por los otros autores; ya que, tienen un *Gap* promedio igual a 0,0588, el cual es bastante pequeño.

Cuadro 4.6: Resultados computacionales del algoritmo *TS-II* con los problemas de B-R-W y comparados con los resultados obtenido por éstos, el mejor resultado obtenido por A-R-P-T, multiLS y LSGA

n	B-R-W	A-R-P-T 100 iter.	A-R-P-T 10000 iter.	multiLS	LSGA	*TS-II*
12	1188.02	1186.81	1186.81	1186.83	1186.81	1232.25
14	1469.19	1467.74	1467.74	1467.76	1467.74	1563.16
16	1476.80	1475.13	1475.13	1475.15	1475.13	1584.14

Capítulo 5

Algoritmo de búsqueda tabú para el $3AP$-planar

5.1. Introducción

En el presente capítulo se presenta una herramienta que permite obtener, de manera rápida y eficiente, sino la óptima, una "buena"solución para el *3AP*-planar; para ello se empleó, al igual que en el Capítulo 4 para el caso axial, la *Búsqueda Tabú* (*TS*), por ser esta una metaheurística diseñada para obtener "buenas" soluciones de problemas de optimización combinatoria.

Como se mencionó con anterioridad, *TS* es una extensión de los métodos de búsqueda local, es por ello que el algoritmo desarrollado presenta una combinación de búsqueda local con memoria a corto plazo y memoria a largo plazo, teniendo como primeros elementos básicos la definición de su espacio de búsqueda y la estructura de la vecindad y como elemento a largo plazo la intensificación de la búsqueda [Gendreau, 2003].

5.2. Metodología

La metodología empleada para realizar el algoritmo que aquí se presenta, es una combinación de la ingeniería del software, mediante el paradigma de ciclo de vida clásico de Pressman (1990) y ciclo de desarrollo de sistemas de información de Fábregas (1991), a través del modelo de desarrollo de software basado en componentes.

La actividad de ingeniería de software comienza con la identificación de clases candidatas. Esto se lleva a cabo examinando los datos que se van a manejar por parte de la aplicación y el algoritmo que se va a aplicar para conseguir el tratamiento. Los datos y los algoritmos correspondientes se empaquetan en una clase. Las clases creadas en los proyectos de ingeniería de software anteriores se almacenan en una biblioteca de clases o diccionario de datos (*repository*). Una vez identificadas las clases candidatas, la biblioteca de clase se examina para determinar si estas ya existen. En caso de que así fuera, se extraen de la biblioteca y se vuelven a utilizar. Si una clase candidata no reside en la biblioteca, se aplican métodos orientados a objetos.

Se compone así la primera iteración de la aplicación a construirse mediante las clases extraídas de la biblioteca y las clases nuevas construidas, para cumplir las necesidades únicas de la aplicación. El modelo de desarrollo basado en componentes conduce a la reutilización del software, produciendo beneficios a los ingenieros de software.

El ciclo de vida clásico, también llamado "modelo secuencial lineal", sugiere un enfoque sistemático y secuencial para el desarrollo del software que comienza en un nivel de sistemas y progresa con el análisis, diseño, codificación, pruebas y mantenimiento.

Modelado, según el ciclo de ingeniería convencional, el modelo lineal secuencial, empleado en esta investigación, comprende las siguientes activi-

dades:

- Análisis de los requerimientos de software: el proceso de reunión de requisitos se intensifica y se centra especialmente en el software. Para comprender la naturaleza del(los) programa(s) a construirse, el análisis del software comprende el dominio de información del software, así como la función requerida, comportamiento, rendimiento e interconexión.

- Diseño: el diseño del software es realmente un proceso de muchos pasos que se centra en cuatro atributos distintos de programa: estructura de datos, arquitectura de software, representaciones de interfaz y detalle procedimental (algoritmo). El proceso del diseño traduce requisitos en una representación del software, es por ello que aquí se diseñaron los procedimientos algorítmicos y las estructuras de datos a utilizar: arreglos tanto lineales como de dos y tres dimensiones. También se seleccionó el software de programación, el cual fue C^{++}.

- Generación de código: el diseño se debe traducir en una forma legible por la máquina, el paso de generación de código lleva a cabo esta tarea. El diseño fue realizado de forma detallada generándose el código, en el lenguaje seleccionado C^{++}, mecánicamente. En esta fase se implementó el algoritmo diseñado en la fase anterior haciendo uso de la programación modular. Para la memoria a corto plazo de TS se empleó la memoria explícita; es decir, se conservaron las soluciones completas y consistió de todas las soluciones visitadas durante la búsqueda. El elemento de memoria a corto plazo empleado fue la estrategia para la lista de candidatos. Debido a que no se busca solucionar un problema determinado, se genera una matriz de costo y en cada ejecución del programa se utilizan valores diferentes en dicha matriz. La solución

inicial se obtiene a partir de un cuadrado latino y de este se genera la vecindad, se intensifica la búsqueda y de ésta se selecciona una nueva solución o un nuevo cuadrado latino, el cual representará la solución actual, luego se obtiene la nueva vecindad donde es buscada la solución actual nueva y así sucesivamente hasta que se cumpla el criterio de parada. Finalmente, la mejor solución es aquella de menor coste de asignación entre todas las visitadas.

- Pruebas: una vez que se ha generado el código, comienzan las pruebas del programa. El proceso de pruebas se centra en los procesos lógicos internos del software, asegurando que todas las sentencias se han comprobado y en los procesos externos funcionales; es decir, realizar las pruebas para la detección de errores y asegurar que la entrada definida produce resultados reales de acuerdo con los resultados requeridos. Además, se realizaron comparaciones de los resultados con los obtenidos por un algoritmo exacto codificado en LINDO.

- Mantenimiento: el software indudablemente sufrirá cambios después de terminado. Se producirán cambios porque se han encontrado errores, debe adaptarse para acoplarse a los cambios de su entorno externo o porque se requieren mejoras funcionales o de rendimiento. El soporte y mantenimiento del software vuelve a aplicar cada una de las fases precedentes a un programa ya existente y no a uno nuevo.

5.3. Algoritmos de búsqueda tabú (TS)

El desarrollo del algoritmo TS para resolver el $3AP$-planar, que se presentan en este capítulo, se llevó a cabo basándose en los elementos básicos de TS como se presenta a continuación.

5.3.1. Espacio de búsqueda y estructura de la vecindad

La selección de un espacio de búsqueda y una estructura de vecindad es el paso más crítico en el plan de cualquier heurístico TS, por tanto es en este paso donde uno debe hacer un mejor uso de la comprensión y conocimiento del problema.

El *espacio de soluciones* de una búsqueda local o de una heurística TS es simplemente el espacio de todas las posibles soluciones que pueden ser consideradas (visitadas) durante la búsqueda. Luego, en el TS adaptado al $3AP$-planar, el espacio de búsqueda considerado es el conjunto de todos los cuadrados latinos asociados al problema, donde cada cuadrado latino del espacio de búsqueda corresponde a un conjunto de ternas que conforman una asignación. Por tanto, el número de elementos del espacio de búsqueda en el algoritmo TS aquí desarrollado es: [Rayser, 1963]

$$L(n,n) := n!(n-1)!l(n,n) \tag{5.1}$$

donde $l(n,n) = l_n$ representa el número de cuadrados latinos reducidos.

En general, no es fácil encontrar todos los cuadrados latinos reducidos de orden n [Rayser, 1963]. En la Tabla 5.1 se muestran algunos valores de l_n.

Cuadro 5.1: Número de cuadrados latinos y cuadrados latinos reducidos para $n \leq 7$

n	*1*	*2*	*3*	*4*	*5*	*6*	*7*
l_n	1	1	1	4	56	9408	16942080
$L(n,n)$	1	2	12	576	161280	812851200	61479419904000

Como puede observarse, el número de cuadrados latinos, y por ende el

tamaño del espacio de búsqueda, crece exponencialmente a medida que n crece.

Estrechamente ligada a la definición del espacio de búsqueda está la de la *estructura de la vecindad*. A cada iteración de la heurística TS, las transformaciones locales que puedan aplicarse a la solución actual (S), definen un conjunto de soluciones vecinas ($N(S)$) en el espacio de búsqueda.

En nuestro algoritmo TS aplicado al $3AP$-planar, a partir de una solución actual o del cuadrado latino actual, se generan las candidatas (vecindad) que son soluciones "cercanas" a la solución actual, éstas se obtienen prefijando una columna e intercambiando todas las demás con ella, luego se fija una fila y se hace el mismo procedimiento, obteniéndose $2n - 1$ cuadrados latinos distintos, aquí se intensifica la búsqueda generándose nuevas vecinas de la solución actual, estableciéndose un radio de acción, el cual consiste en fijar un número de columnas o filas, de acuerdo al caso, e intercambiando las restantes. Es de hacer notar, que a medida que n aumenta, el número de columnas o filas fijas debe aumentar, esto motivado al gran número de cuadrados latinos en el espacio de soluciones, reduciéndose así el tiempo de cómputo; es decir, a medida que el espacio crece, la vecindad a utilizar es cada vez más pequeña, con relación al espacio de soluciones, con el fin de reducir el tiempo de cómputo.

En el algoritmo TS aquí desarrollado se tomó una vecindad consistente de $2n - 1$ cuadrados latinos reducidos. Al intensificar la búsqueda, para $n \leq 7$ se fija una columna o fila, respectivamente, mientras que para $n > 7$, el número de columnas o filas fijas aumenta, evitando así revisar gran número de soluciones.

5.3.2. Estatus tabú

El *Estatus Tabú* es el elemento distintivo de *TS* cuando se compara con búsqueda local. Como se mencionó en la Sección 2.6, el estatus tabú es usado para prevenir el ciclaje al moverse fuera del óptimo local a través de un movimiento de no-mejora. El hecho importante aquí es que cuando esta situación ocurre, se necesita hacer algo para impedir a la búsqueda rastrear sus pasos anteriores que invierta el efecto de recientes movimientos.

En el algoritmo *TS* aquí presentado, se utilizó memoria a corto plazo para tener en cuenta los atributos de soluciones que hayan sido cambiados en el pasado reciente, también se realizó un procedimiento de intensificación, como elemento de memoria a largo plazo, estos junto a una función que verifica si la solución actual pertenece a la historia de movimientos, que evita revisitar soluciones poco atractivas. Los atributos seleccionados que se presentaran en soluciones recientemente visitadas se designan *tabú activos* y las soluciones que tengan ese tipo de elementos se convierten en *tabú*.

5.3.3. Criterio de aspiración

Como se mencionó en el capítulo anterior, el estatus tabú es un elemento importante en un algoritmo *TS*, ya que éste puede prohibir movimientos atractivos, incluso cuando no hay peligro de que se produzca ciclaje, o pueden llevar a un estancamiento global del proceso de búsqueda. Es por ello, que en este algoritmo existe la necesidad de usar dispositivos algoritmicos que permitan la cancelación del estatus tabú. Estos dispositivos son llamados *criterio de aspiración*.

El criterio más simple de aspiración y el más frecuentemente utilizado consiste en permitir un movimiento, aún cuando tenga el status tabú, si éste

produce una solución con un mejor valor objetivo que el de la mejor solución conocida actualmente, partiendo de que la nueva solución no se ha visitado previamente. Este es el criterio de aspiración utilizado en el desarrollo del algoritmo TS presentado en este capítulo.

5.3.4. Criterio de parada

A pesar de que la búsqueda puede seguir indefinidamente, existe la imperiosa necesidad de que ésta tiene que ser detenida en algún punto.

En nuestro algoritmo TS el *criterio de parada* implementado consiste en realizar la búsqueda, a partir de una solución inicial, haciendo uso de los elementos anteriormente descritos hasta agotar el número total de soluciones existentes en la vecindad.

Si una solución es atractiva, se explora por fila y por columna, si se obtiene una "buena"solución se continúa explorando, en caso contrario se etiqueta tabú, si no es atractiva se etiqueta tabú. Si ha sido explorada, tanto por fila y por columna, se etiqueta tabú, de esta forma se exploran todas las soluciones de la vecindad.

5.3.5. Lista de candidatos

El control del número de movimientos examinados se lleva a cabo a través de la estrategia de *lista de candidatos*, que proporciona caminos estratégicos para generar un subconjunto $N'(S) \subseteq N(S)$, el cual se considera útil en la búsqueda de soluciones.

En el algoritmo presentado en este capítulo, haciendo uso de la metaheurísticos TS, se utiliza una estrategia para la lista de candidatos que consiste en restringir el número de soluciones examinadas, disminuyendo la vecindad para los casos en los que el entorno de la solución es grande.

Como se mencionó en la Subsección 5.3.1, para el algoritmo TS desarrollado se consideró una vecindad de $2n-1$ cuadrados latinos reducidos. Al intensificar la búsqueda, para $n \leq 7$ se fija una columna o fila, respectivamente, mientras que para $n > 7$, el número de columnas o filas fijas aumenta, evitando así revisar gran número de soluciones.

5.3.6. Soluciones iniciales

En este algoritmo es utilizada una solución inicial haciendo uso de un cuadrado latino.

Los costes de cada tripleta son generados aleatoriamente y para seleccionar la solución inicial se procedió a construir un cuadrado latino de orden n. En este cuadrado latino no debe repetirse ningún índice (k_i) ni en la fila i ni en la columna j, para toda i y para toda j, con $1 \leq i \leq n$, $1 \leq j \leq n$ y $1 \leq k_i \leq n$.

Debido al tamaño del espacio de soluciones del $3AP$-planar es imposible explorarlo todo a través de cuadrados latinos, por tal razón se debe elegir una solución inicial que permita alcanzar una buena solución para el mismo. Una forma, muy utilizada y sencilla, para construir un cuadrado latino inicial es la siguiente: cada columna j comienza con el número j y continúa las siguientes filas en orden creciente, $j+1$, $j+2$, hasta llegar a n, luego si no se ha alcanzado la última fila, se continúa en las siguientes filas, reiniciando el conteo con un uno (1) en orden creciente hasta llegar a la última fila. En estas condiciones, se puede determinar, en orden, los elementos en una columna j. En general la columna j del cuadrado latino inicial se caracteriza por fila(i) $= i + j - 1$, si $i < n - j + 1$ y fila(i) $= n$, si $i = n - j + 1$.

En estas memorias, considerando el gran número de cuadrados latinos reducidos existentes, se tomaron como base, para la solución inicial, los

cuadrados latinos reducidos asociados a un bloque de restricciones específico, obteniéndose así soluciones iniciales para la exploración de subregiones o áreas de soluciones del 3*AP-planar*.

Existe una correspondencia entre cuadrados latinos asociados a bloques de restricciones del 3*AP-planar* y soluciones del mismo. Tal correspondencia, en cada bloque, entre un cuadrado latino y una solución del 3*AP-planar* es unívoca. Este cuadrado latino reducido inicial no es muy útil a la hora de obtener una buena solución, dado que sólo se estará explorando una misma subregión o área del espacio de soluciones de un instance del 3*AP-planar*.

A continuación se describe el procedimiento utilizado para obtener los cuadrado latinos reducidos iniciales, del algoritmo, conociendo previamente el vector de coste de un instance del 3*AP-planar*.

- Fase I. Inclusión de índices de costes en un rango permitido: en primer lugar, se revisan m costes y se incluyen aquellos índices que satisfagan el rango permitido:

  1.- Construya un vector CO (Coste Ordenado) con los elementos del vector de costes C, del instance del 3*AP-planar*, en orden creciente.

  2.- Seleccione un bloque de restricciones del 3*AP-planar*, considerando una terna de índices (i,j,k) con dos índices fijos y uno variable.

  3.- Establezca un rango de selección de los costes.

  4.- Establezca un número m de coeficientes de costes a incluir, con $1 \leq m \leq n^3$.

  5.- Crear una matriz nula de orden n para construir el cuadrado latino reducido inicial.

6.- Asignar los n primeros números naturales, en orden creciente, en la primera fila y la primera columna.

7.- Mientras el número de costes es menor que m. Seleccione coeficientes de coste en un rango permitido $RangoMin \leq C_{ijk} \leq RangoMax$.

a.- Si los índices fijos son j y k, y el índice libre es i, entonces verifique que el índice i no esté en la fila ni en la columna de la matriz, luego incluya el índice i en la posición (j,k) de la matriz.

b.- Si los índices fijos son i y k, y el índice libre es j, entonces verifique que el índice j no esté en la fila ni en la columna de la matriz, luego incluya el índice j en la posición (i,k) de la matriz.

c.- Si los índices fijos son i y j, y el índice libre es k, entonces verifique que el índice k no esté en la fila ni en la columna de la matriz, luego incluya el índice k en la posición (i,j) de la matriz.

d.- En caso que no se pueda incluir el índice en la matriz, vaya al paso 7.

e.- Fin mientras.

Si la matriz está completa, es decir, si satisface las condiciones de los cuadrados latinos reducidos, entonces el procedimiento termina. En caso contrario, se debe ir a la Fase II.

- Fase II. Completación del cuadrado latino reducido: en esta fase se procederá a incluir índices que permitan la construcción de un cuadrado latino, sin importar la condición de rango permitido.

Inicialmente, se construye una matriz de orden $n-2$ para guardar los índices candidatos de cada posición (fila, columna) del cuadrado latino, esto incluye los índices que ocupan la primera fila y primera columna. Después, se construyen dos conjuntos de índices para guardar los índices que están en una columna y en una fila determinada. Esto se hace con la finalidad de cruzar la información de los índices que están y los que no pueden estar, tanto en la fila, como en la columna de la matriz.

1.- Se recorre la matriz **A** de cuadrados latinos por fila, en orden creciente.

2.- Si existe un candidato r a ocupar la posición (i,j) y no está en la fila i, entonces incluir en esta posición. Luego se excluye el índice de la lista de candidatos y se elimina de la columna j en caso de que esté incluido.

3.- Si no existe un candidato r a ocupar la posición (i,j), entonces se busca un candidato s por columna a ocupar la posición (i,j). Si s no está en la columna, entonces incluirlo en esta posición. Luego se excluye el índice de la lista candidatos y se elimina de la fila i, en caso de que esté incluido.

4.- Se continúa con los pasos 2 y 3 hasta recorrer todas las filas de **A**, completando de esta forma el cuadrado latino o con la respuesta de que no se ha podido completar el cuadrado latino, para los costes especificados.

5.3.7. Estrategia de intensificación

Dos componentes altamente importantes de largo plazo de TS son las estrategias de intensificación y de diversificación. Las estrategias de intensi-

ficación, las cuales son las que nos interesan en este trabajo, están basadas en la modificación de reglas de elección, de tal forma que se favorezcan combinaciones de movimientos y características de solución que históricamente hayan sido "buenas". Pueden además iniciar un regreso hacia regiones atractivas para buscar en ellas más extensamente.

Considérese una zona τ y una vecindad $N(\boldsymbol{x})$ de la región factible, donde una *zona* es una porción de la región factible del $3AP$-*planar* que contiene soluciones que están asociadas a cuadrados latinos con la n-ésima columna igual. La solución $\boldsymbol{x}$ tiene asociado un cuadrado latino, el cual tiene las columnas n y $n-1$ fijas. Se establece la siguiente estrategia de intensificación en $N(\boldsymbol{x})$:

- Sea r un contador que indique el número de columnas fijas adicionales en el cuadrado latino asociado a la vecindad $N(\boldsymbol{x})$ y $n-(r+1)$ la posición de la nueva columna que se fija en el cuadrado latino.
- Al principio se toma $r=1$ y se fija una columna en la posición $n-2$, realizándose movimientos con a_{n-2} y cada una de las restantes $n-3$ columna libres.
- Seguidamente, se aumenta r en una unidad y se fija ahora la posición $n-3$, realizándose movimientos con a_{n-3} y cada una de las restantes $n-4$ columnas libres.
- Este procedimiento continúa hasta $r=n-1$, se fija la posición 2 y se realiza el intercambio entre las dos últimas columnas en el cuadrado latino.

Definición 5.3.1 *Se llama transposición columnas (filas) de un cuadrado latino* $\boldsymbol{A}$*, a toda permutación columnas (filas) de* $\boldsymbol{A}$ *que permute solamente las columnas (filas)* a_i *y* a_j *(*a^i *y* a^j*).*

Luego para el almacenamiento de soluciones de calidad de una vecindad, se procede de la siguiente forma: sea i el índice de una columna que se fijará en el cuadrado latino $\mathbf{A}$ asociado a $\boldsymbol{x}$ de una vecindad $N(\boldsymbol{x})$. Por cada columna libre que sea ubicada en esta posición, se determina una sucesión de transposiciones del tipo $A_{(i,j)}$. Para simplificar los cálculos, se tomará la mejor (las mejores en caso de empate) por cada columna i fija. Con lo cual se construye una lista de soluciones de calidad contenidas en la vecindad.

5.3.8. Algoritmo

En primer lugar, considérese un instance del $3AP$-*planar* asociado a tres n-conjuntos I, J, $K \subset N$ y un cuadrado latino de orden n asociado a una solución del instance. Motivado a la imposibilidad de explorar toda la región factible, partiendo de un cuadrado latino inicial, y además, el intercambio de filas o columnas en un cuadrado latino es un problema de tipo exponencial, se plantea una serie de estrategias que permitan hallar "buenas"soluciones, en un tiempo razonable, mediante la metaheurística búsqueda tabú:

- **Agrupar las restricciones del $3AP$-*planar* en bloques**. Se fijan dos índices y se deja un índice libre. Luego se construyen tres bloques de restricciones que contienen exactamente, cada uno, las n^3 variables del instance, permitiendo la exploración de soluciones a través de planos paralelos entre si y perpendiculares a un eje coordenadas de $\mathbb{R}^3$. Así, la búsqueda es realizada en un bloque de restricciones y en caso de ser necesario, se exploran todos los bloques, con el fin de seleccionar la mejor solución.

- **Construcción de un cuadrado latino reducido de bajos costes**. Como el problema de determinar el número de cuadrados latinos de

orden n es de tipo exponencial y este número también es difícil de obtener, entonces se dificulta la exploración de la región factible mediante la intensificación. Por tal razón, se plantea la construcción de un cuadrado latino que contenga inicialmente la mayor cantidad de índices asociados a bajos costes de la función objetivo. Esto permite iniciar el algoritmo de búsqueda tabú con una solución que no esté muy alejada de la solución óptima.

- **Determinar zonas de exploración en el espacio de soluciones**. Las zonas están asociadas a cuadrados latinos que tienen fija la última columna. Toda vecindad en una zona satisface la característica de dicha zona. Partiendo de un cuadrado latino inicial se obtienen, al comienzo, n zonas de exploración, cada zona contiene vecindades cuyo radio depende del número de columnas que se vayan fijando a partir de la n-ésima columna y cada solución en la vecindad mantiene las características de la vecindad y de la zona. También es posible obtener nuevas zonas mediante el intercambio de filas, en cada cuadrado latino.

- **Crear lista de soluciones tabú**. Cada nueva solución obtenida mediante la transposición de fila y/o columna es incluida en una lista de *soluciones exploradas*. Cuando una solución es explorada, tanto por filas como por columnas, es incluida en una lista de soluciones tabú y eliminada de la lista soluciones exploradas, esto se hace para evitar la exploración de soluciones ya visitadas. Un criterio de parada para el algoritmo se dará cuando la lista tabú contenga todas las soluciones exploradas y la lista soluciones en exploración esté vacía.

- **Crear lista de buenas soluciones**. A medida que se explora el espacio de soluciones, se construye una lista de "buenas"soluciones, en orden

creciente. Ante la posibilidad de que hayan quedado algunas soluciones sin explorar y que conduzcan a una mejor solución. Por esta razón, son guardadas para realizar nuevas exploraciones alrededor de ellas y así proseguir con la búsqueda de forma intensa.

- **Intensificación de la búsqueda**. De todas las "buenas"soluciones encontradas, se toma un número determinado de ellas y se intensifica la búsqueda en sus entornos y se prosigue en la búsqueda de una mejor solución. El proceso de exploración termina cuando se hayan revisado todas las buenas soluciones seleccionadas y no se tengan nuevas soluciones para explorar.

Partiendo de las estrategias antes señaladas, se realiza el diseño del algoritmo de búsqueda tabú para el 3AP-*planar*, para ello se presentan a continuación dos pseudo-códigos: uno para encontrar un cuadrado latino reducido de bajo coste para el 3AP-*planar* y el otro el algoritmo en sí para hallar soluciones del 3AP-*planar* a través de una búsqueda tabú.

Pseudocódigo para encontrar un cuadrado latino reducido de bajo coste para el 3*AP-planar*.

1.- Inicio

2.- Leer n

3.- Leer un instance del 3AP-*planar*

4.- Seleccionar un bloque de restricciones

5.- Definir una matriz de orden n para el cuadrado latino

6.- Insertar índices, en la matriz, en orden creciente, tanto en la primera fila como en la primera columna y en el resto del arreglo inserte cero

7.- Definir un vector de orden $m = n^3$

8.- Insertar en el vector los índices asociados a los costes que llevan un orden creciente

9.- Colocar los índices asociados a los menores costes, tanto como sea posible, manteniendo las características de los cuadrados latinos

10.- Si no hay ceros en la matriz, entonces se ha obtenido el cuadrado latino y fin del algoritmo. Si no, se aplican los siguientes pasos:

11.- Definir un vector de índices Tabú de orden n

12.- Definir un vector de índices Candidatos de orden n

13.- Mientras existan ceros en la matriz

  a.- Buscar ceros (j,k)

  b.- Si hay cambio de fila, reiniciar vector Tabú

  c.- Revisar la fila j y la columna k hasta la fila $j-1$ de la matriz y actualizar Candidatos(k). Vaya al paso (d)

  d.- Si Candidatos(k) $\cap$ Tabu(k) =1, entonces asignar a la posición (j,k) de la matriz y actualizar vector Tabú. Si no, revisar la fila j hasta la columna k - 1 y la columna k hasta la fila j - 1 y actualizar Candidatos(k). Vaya al paso (e)

  e.- Si Candidatos(k) $\cap$ Tabu(k) =1, entonces asignarlo a la posición (j,k) de la matriz y actualizar el vector Tabú. En caso de que el índice esté en la columna k a partir de la fila $j+1$, se asigna cero a esta posición

f.- Si Candidatos(k) $\cap$ Tabu(k) > 1, escoger un índice asociado al menor coste y asignarlo a la posición (j,k) de la matriz y actualizar el vector Tabú

14.- Fin mientras

15.- Fin algoritmo

Pseudocódigo del algoritmo de búsqueda tabú para el 3*AP-planar*.

A continuación se muestra el pseudocódigo del algoritmo de búsqueda tabú para buscar "buenas"soluciones para el 3*AP-planar*.

1.- Inicio

2.- Leer n

3.- Leer un instance del 3*AP-planar*

4.- Generar un cuadrado latino inicial **A** de orden n

5.- Asociar el cuadrado latino inicial a un bloque de restricciones del instance

6.- Crear una lista de Soluciones no Exploradas

7.- Crear una lista de Soluciones Tabú

8.- Crear una lista de Buenas Soluciones

9.- Mientras existan soluciones no exploradas:

a.- Fijar una zona en el cuadrado latino asociado a la solución no explorada

b.- Si la exploraciones es por columna, entonces:

i.- Realizar transposiciones de columnas no fijas en el cuadrado latino

ii.- Anexar las mejores soluciones a la lista ordenada de Buenas Soluciones

iii.- Incluir las soluciones exploradas por filas y por columnas en la lista Soluciones Tabú y excluirlas de la lista Soluciones no Exploradas

c.- Fin si

d.- Si la exploraciones es por filas, entonces

i.- Realizar transposiciones de filas no fijas en el cuadrado latino

ii.- Anexar las mejores soluciones a la lista ordenada de Buenas Soluciones

iii.- Incluir las soluciones exploradas por filas y por columnas en la lista Soluciones Tabú y excluirlas de la lista Soluciones no Exploradas

e.- Fin si

10.- Fin mientras

11.- Seleccionar las mejores soluciones encontradas de la lista Buenas Soluciones

12.- Intensificar la búsqueda, explorar las mejores soluciones

13.- Escribir el mejor valor objetivo encontrado y la solución asociada a él

14.- Fin

5.4. Resultados computacionales

En esta sección se construyen aleatoriamente instances del $3AP$-*planar*, para probar los algoritmos de generación de cuadrados latinos reducidos de bajo coste y de búsqueda tabú, descritos en la sección anterior y que fueron implementados en C^{++}. Además, se realizarán comparaciones entre los resultados obtenidos por los algoritmos diseñados y los producidos haciendo uso del paquete LINDO.

Es importante resaltar, que a medida que n aumenta el tamaño del instance aumenta significativamente, en cuanto al número de variables y de restricciones, como se muestra en la Tabla 5.2, influyendo esto notablemente en el tiempo computacional empleado en la búsqueda de soluciones de un instance y además, empíricamente se puede ver, que la complejidad algorítmica del $3AP$-*planar* es exponencial.

En la búsqueda de soluciones, se genera un cuadrado latino inicial, utilizando los costos de la función objetivo en un rango permitido que está en el intervalo $[0, 1]$.

En la exploración del espacio de soluciones para $n = 2$ y 3, sólo se necesita un cuadrado latino reducido inicial, mientras que para $n = 4$, se determinaron los cuatro (4) cuadrados latinos reducidos, que generan todos los cuadrados latinos de orden 4.

Ahora, a partir de $n = 5$, se generan cuadrados latinos iniciales utilizando los costos de la función objetivo del instance. Para generarlos, se toma un rango que está en el intervalo $[0, 1]$. Por ejemplo, si el rango mínimo es cero (0), se toma en cuenta los costes más altos y si el rango máximo es uno (1), se consideran los costes más pequeños.

Motivado al hecho que para $n = 5$ el número de cuadrados latinos reducidos es de cincuenta y seis (56) y el número de cuadrados latinos es de

Cuadro 5.2: Número de variables vs. tamaño del instance

n	Número de variables (n^3)	Número de restricciones ($3n^2$)
2	8	12
3	27	27
4	64	32
5	125	75
9	729	243
10	1000	300
15	3375	675
20	8000	1200
25	15625	1875
30	27000	2700
40	64000	4800

quinientos setenta y seis (576), lo cual aumenta de considerablemente a medida que aumenta el valor de n [Sloane, 2003], entonces a partir de $n = 5$, se hace la búsqueda desde una clase de cuadrados latinos, donde el cuadrado latino reducido generado es su representante.

En la Tabla 5.3, se muestran los resultados computacionales obtenidos al resolver los problemas generados para el $3AP$-*planar* para $n = 2, 3, 4, 5, 6, 10, 20$ y 40. Es importante resaltar, que se generaron cinco (5) instances para cada valor de n y se muestran los promedios de los resultados obtenidos para cada n.

Es de hacer notar, que en dicha tabla se muestra los resultados haciendo uso del paquete computacional LINDO (valor óptimo, tiempo de resolución y número de iteraciones realizadas) y se comparan con los resultados obtenidos

al aplicar el algoritmo desarrollado (valor, tiempo de resolución y valor de la función *Gap*).

Cuadro 5.3: Resultados computacionales del algoritmo *TS* para el 3*AP-planar*

n	*LINGO*			*TS*		
	valor	t.(seg.)	num. iter.	valor	t.(seg.)	*Gap*
2	**166.6**	**0.0000**	3	**166.6**	**0.0000**	0.00
3	**387.6**	**0.0000**	26	**387.6**	**0.0000**	0.00
4	**473.2**	**0.0000**	43	**473.2**	0.0872	0.00
5	**708.2**	**0.0000**	193	799,8	2.747	0.13
6	**1035**	**0.0000**	1151	1136.8	9.946	0.098
10	—	—	—	3650.2	12.975	—
20	—	—	—	17476.2	14.662	—
40	—	—	—	75566.2	103.871	—

Se puede observar, en la Tabla 5.3, que para $n = 2$ y 3, el algoritmo desarrollado obtuvo el valor óptimo y un tiempo de cero segundos (0.00 seg.), para $n = 4$ también se obtuvo el valor óptimo, pero con un tiempo de 0.0872 seg. Para los valores de $n \geq 5$, no se obtuvo el valor óptimo, sin embargo, los resultados obtenidos están muy cercanos al óptimo, como se puede ver para $n = 5$ y 6, donde los valores de la función *Gap* son de 0.13 y 0.098, respectivamente.

Por lo anteriormente expuesto, podemos suponer que para $n \geq 5$ el algoritmo propuesto produce “buenosresultados.

Capítulo 6

Algoritmo sistema hormiga para el $4AP$

6.1. Introducción

En este capítulo proponemos un procedimiento haciendo uso de la metaheurística "sistema hormiga"para encontrar la mejor solución al $4AP$. Su efectividad se evalua sobre problemas generados aleatoriamente.

En el $3AP$, como se planteó anteriormente, se distinguen dos tipos de problemas, el $3AP$-axial y el $3AP$-planar. En el caso del $4AP$, también podríamos hablar de varios tipos de problemas, fijando, en cada restricción, solo uno de los subíndices, fijando dos subíndices o fijando tres subíndices, que utilizando las definiciones dadas en el $3AP$, podríamos hablar del axial, planar y espacial, respectivamente.

En este capítulo y en el trabajo en general, trataremos sólo el $4AP$ cuando se fija un sólo subíndice en cada restricción y hablaremos de él como el $4APA$.

Se han diseñado métodos de resolución para este tipo de problemas, los cuales aunque no obtengan la solución óptima, producen una "buena" aprox-

imación a ésta. Entre estos métodos están las metaheurísticas, siendo una de ellas el sistema hormiga.

Se han propuesto muchos algoritmos basados en sistema hormiga para solucionar diversos tipos de problemas de optimización combinatoria, algunos de estos problemas y autores son: el problema del agente viajero, en investigaciones realizadas por: Dorigo et al. (1991), Gambardella y Dorigo (1995), Dorigo y Gambardella (1997) Stutzle y Hoos (1997) y Bullnheimer et al. (1997), asignación cuadrática, en trabajos realizados por: Maniezzo et al. (1994), Gambardella (1997), Taillard y Dorigo (1998) Stutzle y Hoos (1998) y Maniezzo y Colorni (1998), ruta de vehículos, investigación de: Colorni et al.(1994), coloreado de un grafo, Costa y Hertz (1997), entre otros. También se han desarrollado algoritmos heurísticos y/o metaheurísticos para resolver el mAP, para m:= 3, 4, 5, 6, 7 y 8, tales como los desarrollados por: Oliveira y Pardalos (2004), desarrollan algoritmos heurísticos paralelos aleatorizados para m:= 5, 6 y 7; Gutin y Karapetyan (2009a) presentan una heurística de búsqueda local para m:= 3, 4, 5, 6, 7 y 8; a su vez, Gutin y Karapetyan (2009b) desarrollan un algoritmo memético para m:= 3, 4, 5 y 6; y finalmente Karapetyan y Gutin (2010) resuelven problemas de asignación multidimensional para m:= 3, 4, 5 y 6, pero hasta ahora no se ha propuesto ningún algoritmo basado en sistema hormiga para resolver el mAP.

6.2. Metodología

Para el desarrollo de la implementación de esta investigación, se utilizó una metodología híbrida entre la metodología de investigación de operaciones descrita por Taha (1991) y ciclo de vida de desarrollo de software (Pressman,2002), la cual está constituida por las etapas que se describen a

continuación.

Definición del problema: aquí se realizó una descripción del problema en estudio, definiéndose las variables de decisión y sus limitaciones, así como los objetivos.

Construcción del modelo: en esta fase se representó el problema en expresiones matemáticas, definiéndose la función objetivo, restricciones y tipo de variables de decisión, que serán empleados.

Fase de diseño: es realmente un proceso multipaso que se enfoca sobre cuatro atributos distintos del programa para resolver el problema: la estructura de datos, caracterización de la interfaz, el detalle procedimental y funcional. Todo este diseño se llevó a cabo tomando en cuenta un esquema modular, que satisfaga los requerimientos del software de la implementación antes mencionada.

Fase de codificación: en esta fase se escogió el lenguajes de programación C++. Este lenguaje facilita la programación modular gracias a la inclusión de las estructuras de Clases. Además, este lenguaje incluye funciones que facilitan la manipulación de los datos, lo que permite una reducción importante del tiempo de ejecución. Luego se codificó el algoritmo planteado en el lenguaje seleccionado.

Implementación del modelo: la manipulación de los parámetros a utilizar permitió obtener las soluciones del modelo.

Validación del modelo: se comprobó la confiabilidad del funcionamiento del modelo a través de los resultados obtenidos en la fase anterior, para ello se utilizó la técnica de la media muestral para cada N, comparando los resultados obtenidos por el algoritmo desarrollado con los arrojados por el algoritmo exacto codificado en XPRESS.

6.3. Algoritmo sistema hormiga

El algoritmo que se define es un modelo derivado del estudio de colonias de hormiga aplicado a problemas de optimización combinatoria, llamado Sistema Hormiga (*Ant System(AS)*). La idea básica del algoritmo sistema hormiga, es generar una gran cantidad de agentes artificiales simples, para poder construir "buenas"soluciones a los problemas de optimización combinatoria. Este sistema tendrá algunas diferencias comparado con el real (natural): aquí las hormigas artificiales tienen memoria, no son completamente ciegas y viven en un ambiente donde el tiempo es discreto.

6.3.1. Descripción general del proceso

En primer lugar se inicializa el rastro de cada una de las acciones posibles. A continuación, cada una de las hormigas construye una solución escogiendo de una lista de nodos los posibles. La elección del nodo se realiza tomando en consideración la información suministrada por la metaheurística y el rastro dejado por las hormigas en las iteraciones previas. El nodo escogido se almacena en una lista tabú para guardar los nodos visitados en tiempo t y prohíbe a la hormiga visitarlos de nuevo antes de la n-ésima iteración. Estos dos pasos se reiteran hasta que la hormiga obtiene una solución factible del problema. Para que esta solución sea candidata a ser la mejor, va a depender de la actualización del rastro de feromona de la hormiga asociada. Para este proceso existen tres variantes: (1) hormiga ciclo: en este caso, las hormigas, una vez finalizado su recorrido, segrega una cantidad de feromona que es inversamente proporcional a la longitud del recorrido. (2) Hormiga densidad: en esta variante, cada hormiga segrega una cantidad fija de feromona durante cada recorrido por los nodos. (3) Hormiga cantidad: cada hormiga

segrega feromona durante cada nodo visitado, pero segrega una cantidad que es inversamente proporcional a la distancia entre los nodos.

Cuando todas las hormigas concluyen esta fase constructiva, los incrementos parciales que se han obtenido permiten actualizar el rastro de feromona. Este proceso se repite hasta satisfacer un criterio de parada. La mejor solución, obtenida durante la ejecución del algoritmo, es la solución propuesta por el procedimiento.

6.3.2. Algoritmo

En este trabajo se desarrolló un algoritmo de asignación 4-dimensional y un algoritmo que hace uso de la metaheurística AS, adaptado al algoritmo anterior, denominado $4APAS$, que se describe a continuación.

La fase de inicialización consiste en asignar un valor constante f a la intensidad de feromona (número entero cualquiera) de cada asignación i,j,k,l, en el tiempo cero; es decir $F_{i,j,k,l}(0):=f$. Se toma el incremento total de feromona en i,j,k,l igual a cero, $\Delta F_{i,j,k,l}:=0$. Se procede a encontrar la solución inicial (utilizando un algoritmo greedy). Continuando, se ubica aleatoriamente una hormiga en nodos diferentes representando los nodos de arranque, describiéndose en las matrices: $M_{i[h,r]}$, $M_{j[h,r]}$, $M_{k[h,r]}$, $M_{l[h,r]}$, donde cada uno de los elementos de estas matrices representa una componente de cada 4-upla solución, para la hormiga h-ésima, en la r-ésima iteración-.

Luego de varias pruebas, se decidió, por obtenerse mejores resultados, que el algoritmo trabajara con veinte (20) hormigas (ya que con más veinte (20), se genera un tiempo computacional muy elevado), permitiendo así mayor exploración de los nodos que pueden formar parte de la solución del problema; y un número de quince (15) ciclos que significan quince (15) intentos que van a tener las veinte (20) hormigas para encontrar una buena solución. A

continuación, se asignan valores a los siguientes parámetros: α, que indica la importancia relativa del rastro de feromona, número real tal que $0 < \alpha \leq 1$. β es un parámetro que indica la importancia relativa de la visibilidad. β es un número real tal que $0 < \beta \leq 1$. Q valor constante que indica el rastro de feromona que recibe cada asignación por ser visitada. Q es un número entero positivo cualquiera y ρ representa la cantidad de feromona que desaparece de una asignación por efecto de la evaporación, número entero tal que $0 < \rho \leq 1$.

La siguiente fase es la de construcción y expresa que, mientras queden nodos por visitar, cada hormiga elige aleatoriamente el próximo nodo. Para un i,j,k,l específico se verifica, en cada índice, si ha sido visitado. Si se ha seleccionado se descarta dicha asignación, escogiéndose aleatoriamente una nueva; si no, se escoge el próximo nodo a visitar, esto se hace según la siguiente función de probabilidad, que representa el nivel de intensidad de la feromona:

$$P^h_{i,j,k,l}(t) := \begin{cases} \frac{[F_{ijkl}]^\alpha [\eta_{ijkl}]^\beta}{\sum [F_{idwz}(t)]^\alpha [\eta_{idwz}]^\beta} & , \text{ si } \; d \in S_{J(i)_h}, w \in S_{K(i)_h}, z \in S_{L(i)_h}, \\ & \quad \forall i := 1, \ldots, n. \\ 0 & , \quad \text{en otro caso} \end{cases} \tag{6.1}$$

donde $S_{J(i)_h}$, $S_{K(i)_h}$ y $S_{L(i)_h}$ representan cada uno de los conjuntos de todas las asignaciones j, k y l, respectivamente, tomadas por la h-ésima hormiga, partiendo de i y η^β_{ijkl} determina, que mientras más pequeño sea el coste de una 4-upla, más deseable es esta.

Para la selección de una asignación se escoge un número aleatorio, y si éste es igual o aproximado a una determinada probabilidad acumulada, se escoge la asignación correspondiente a dicha probabilidad, guardándose en una lista para cada hormiga.

Una vez que no existen más nodo a visitar, entonces cada hormiga ha completado un camino; y cuando las 20 hormigas han construido su camino se ha completado un ciclo. A continuación comienza el cálculo del incremento del rastro de feromona; obteniéndose la longitud del camino construido por cada hormiga; luego se calcula el incremento de feromona para la asignación (i, j, k, l) de la h-ésima hormiga, aplicando el método hormiga-ciclo, que ha mostrado buenos resultados en trabajos anteriores [Babaoglu, 1996]. Este método consiste, en que las hormigas segregan la feromona una vez construida la solución; es decir, una vez finalizado su recorrido, segrega una cantidad de feromona que es inversamente proporcional a la longitud del recorrido y se expresa a través de la función $\Delta F^h_{i,j,k,l}$, definida por:

$$\Delta F^h_{i,j,k,l} := \begin{cases} \frac{Q}{Long_h} & , \text{ si la asignación}(i,j,k,l)\text{pertenece al camino} \\ & \text{construido por la } h\text{-ésima hormiga} \\ 0 & , \text{ en otro caso} \end{cases} \tag{6.2}$$

A continuación se calcula el incremento total de feromona para la asignación i,j,k,l, debido a la aportación de todas las hormigas, según la expresión:

$$\Delta F_{i,j,k,l} := \sum_{h=1}^{n} \Delta F^h_{i,j,k,l},$$

la sumatoria se refiere a la suma de los incrementos parciales debido a cada hormiga, para una asignación (i, j, k, l) determinada.

La siguiente fase es la de actualización de feromona, donde se obtiene el nuevo rastro de feromona para las asignaciones (i, j, k, l). Esto se realiza mediante la siguiente fórmula:

$$F_{i,j,k,l}(t + n) := \rho F_{i,j,k,l}(t) + \Delta F_{i,j,k,l},$$

es decir, la feromona actual ($F_{i,j,k,l}(t+n)$), de la cual la asignación (i,j,k,l) ha experimentado un cambio con respecto a la anterior feromona en función de una constante ρ, dada por el usuario ($\rho F_{i,j,k,l}(t)$), más el aporte de las hormigas que han escogido esta asignación ($\Delta F_{i,j,k,l}$). La idea de utilizar el parámetro ρ radica en hacer que las hormigas olviden asignaciones (i,j,k,l) que pudieron ser malas. Así, si una asignación ya no es escogida por las hormigas, la cantidad de feromona irá decreciendo de manera que cada vez tendrá menor probabilidad de ser elegido, este proceso es llamado conducta de estancamiento.

La última fase radica en verificar el criterio de parada; el cual consiste en parar el algoritmo cuando se alcance el número máximo de ciclos, que es igual a catorce (14); ya que para valores mayores el tiempo computacional es muy grande, y no exista conducta de estancamiento; si no se cumple se actualiza el camino más corto encontrado por las veinte (20) hormigas, guardándose en los vectores $A_i[r]$, $A_j[r]$, $A_k[r]$, $A_l[r]$; que representan el valor de la asignación i,j,k y l, respectivamente, que forma parte de la solución del problema en la r-ésima asignación, donde r es el número de iteraciones o de ciclos, volviendo luego a la fase de construcción. Si se cumple el criterio de parada se imprime la mejor solución encontrada y el tiempo empleado.

6.4. Resultados computacionales

Los experimentos se realizaron en una computadora con las siguientes características: procesador Pentium IV, 1.6 GHz, 256 Mb RAM.

Se utilizó el *XPRESS*, el cual es un software de programación lineal para resolver modelos matemáticos, para resolver el $4AP$ para n:=2,3,4,6; ya que fue lo que la versión encontrada del programa permitió. Estos resultados

se compararon con los emitidos por el $4APAS$, que inclusive resolvió casos para n entre 8 y 25. El comportamiento del $4APAS$ en comparación con el *XPRESS*, utilizando los siguientes valores para los parámetros α :=0.8, β :=0.5, *feromona*:=10, Q=1000, ρ :=0.5, es mostrado en la Tabla 6.1, el cual contiene los promedios, entre los cinco problemas generados aleatoriamente para cada n, de los costes y los tiempos obtenidos por el *XPRESS* y el $4APAS$.

Cuadro 6.1: Resultados computacionales obtenidos por el $4APAS$.

n	*XPRESS*	$4APAS$	*gap*	*Tiempo(seg.)*
2	**54.0**	**54.0**	0	0.0
3	**35.0**	**35.0**	0	0.0
4	**19.2**	**19.2**	0	0.0
6	**17.8**	**17.8**	0	0.0
8	—	12.6	—	1.2
10	—	14.2	—	3.2
12	—	19.2	—	6.0
16	—	19.6	—	20.6
20	—	20.8	—	57.2
25	—	29.6	—	176.2

En la Tabla 6.1, puede observarse como el $4APAS$ alcanza la solución óptima para $2 \leq n \leq 6$, y para $8 \leq n \leq 25$, con el $4APAS$ se obtienen valores pequeños, lo que induce a pensar que son soluciones próximas al óptimo. Además, en la Tabla 6.1 puede observarse que el tiempo de cómputo es también pequeño.

Conclusiones

Realizado el trabajo de investigación presentado en esta memoria llegamos a las siguientes conclusiones:

- Los algoritmos metaheurísticos son "buenos" para la resolución de problemas *NP*-difíciles, como en este caso, problemas de asignación multidimensional (mAP), para $m = 3$ y 4.
- Las metaheurísticas algoritmo genético y búsqueda tabú, produjeron "buenos" resultados para el *3AP*-axial, para los problemas tratados en esta memoria.
- Los resultados obtenidos por el algoritmo *TS-I*, para el *3AP*-axial, mejoraron los resultados producidos por el algoritmo genético I, en los problemas tratados.
- Los resultados producidos por el algoritmo genético II mejoraron los resultados de Balas y Saltzman (1991) y en los de otros autores, aunque no los mejoró, los resultados son muy próximos y con tiempo de cómputo muy pequeño.
- Los resultados obtenidos por el algoritmo *TS-II*, al resolver los problemas de la literatura presentados en estas memorias, mejoran en varios casos los existentes en los artículos tratados y en otros, aunque no los mejoran, están muy próximos, con un *Gap* y tiempo de cómputo bastante pequeño.

- Para los problemas de Balas y Saltzman (1991), el algoritmo genético II obtiene mejores resultados que el algoritmo *TS-II*, mientras que para los otros problemas tratados, los resultados obtenidos por el algoritmo *TS-II* son mejores que los alcanzados por el genético II.

- La búsqueda tabú produjo "buenos" resultados, para el *3AP*-planar, para los problemas generados y presentados en estas memorias.

- Aunque el *3AP*-axial y el *3AP*-planar son ambos problemas del tipo *NP*-difíciles, se puede considerar al *3AP*-planar de mayor complejidad para su resolución, ya que tiene un espacio de soluciones mucho mayor al espacio de soluciones del *3AP*-axial, lo cual complica la búsqueda.

- La metaheurística sistema hormiga produjo "buenos" resultados, para el *4AP*, para los problemas tratados en estas memorias.

Bibliografía

[Abellanas y Lodares, 1990] M. Abellanas & D. Lodares. *Análisis de algoritmos y Teoría de Grafos*, RA-MA Editorial, Madrid-España, 1990.

[Aiex et al., 2001] R. Aiex, M. Resende, P. Pardalos & G. Toraldo. Grasp with path relinking for the three-index assignment problem, *INFORMS Journal on Computing,* **17** (2), 2005: 224-247.

[Almirón et al., 1998] M. Almirón, E. Chaparro & B. Barán. Optimización basada en sistema de hormigas con heurística de inicialización, IX Panel de Informática Expomática'98, Asunción-Paraguay, 1998.

[Babaoglu, 1996] O. Babaoglu. The ant system, optimization by a colony of cooperating agents. *Ant Colony Optimization.* <www.cs.unibo.it/babaoglu/courses/cas/tutorials/AntColonyOptimization.pdf>. 1996. (2 de febrero de 2002).

[Balas y Qi, 1993] E. Balas & L. Qi. Linear time separation algorithms for the three-index assignment polytope, *Discrete Applied Mathematics* **43**, 1993: 1-12.

[Balas y Saltzman, 1989] E. Balas & M. Saltzman. Facets of the Three-Index Assignment Polytope, *Discrete Appl. Math.* **23**, 1989: 201-229

[Balas y Saltzman, 1991] E. Balas & M. Saltzman. An algorithm for the three-index assignment Problem, *Oper. Res.* **39**, 1991: 150-161.

[Bandelt et al., 1994] H. Bandelt, Y. Crama & F. Spieksma. Approximation algorithms for multidimensional assignment problems with decomposable costs, *Discrete Appl. Math.* **49**, 1994: 25-50.

[Beasly, 1993] J. E. Beasly. *Modern Heuristic Techniques for Combinatorial Problem. Charter 6 (Lagrangean Relaxation)*, Editado por C. R. Reeves, London, 1993.

[Bondy y Murty, 1979] J. Bondy & V. Murty. *Graph Theory and Related Topics*, Academic Press, New York, 1979.

[Bower, 1994] R. Bower. *Genetics Algoritms and Investment Strategies*, John Wiley & Sons, 1994.

[Brassard y Bartley, 1997] G. Brassard & T. Bartley. *Fundamentos de ALgoritmia*, Editorial Prentice Hall, Madrid, España, 1997.

[Burkard y Çela, 1998] R. Burkard & E. Çela. *Linear Assignment Problems and Extensions*, Optimierung and Kontrollie, <http://citeseer.nj.nec.com/cachedpage/168640>, 1998.

[Burkard et al., 1996a] R. Burkard, B. Klinz & R. Rudolf. Perspectives of Monge properties in optimization, *Discrete Appl. Math.* **70**, 1996: 95-161.

[Burkard et al., 1993] R. Burkard & R. Rudolf. Computational investigations on 3-dimensional axial assignment problems, *Belgian J. Oper. Res. Statist. Comput. Sci.* **32**, 1993: 85-98.

[Burkard et al., 1996b] R. Burkard, R. Rudolf & G. Woeginger. Three dimensional axial assignment problems with decomposible cost coefficients, *Discrete Appl. Math.* **65**, 1996: 65-123.

[Centeno, 2001] M. Centeno. Búsqueda de soluciones para el problema de asignación 3-dimensional axial a través de heurísticas, Trabajo de Ascenso Dpto. de Matemáticas UDO, 2001.

[Centeno, 2007] M. Centeno. Búsqueda de soluciones para el 3AP-axial usando búsqueda por entornos. *FARAUTE* **2** (1), 2007: 20-27.

[Centeno y Salazar, 2008] M.V. Centeno & H. Salazar. Algoritmo para el 4AP haciendo uso de la Metaheurística Sistema Hormiga. *Rev. Ing. Industrial* **7** (2), 2008: 73-83.

[Centeno, Urbina y Salazar, 2010] M.V. Centeno, J. Urbina & J.J. Salazar Un algoritmo de búsqueda tabú para resolver el problema de asignación 3-dimensional axial. *ICHIO* **1** (1), 2010: 35-45.

[Cerny, 1985] V. Cerny. Thermodynamical approach to the Travelling Salesman Problem. An Efficient Simulated Algorithm, *J. of Optimization Theory an Aplications* **45**, 1985: 41-45.

[Cook et al., 1998] W. Cook, W. Cunningham, W. Pulleyblank & A. Schrijver. *Combinatorial Optimization*, John Wiley & Sons, Canadá, 1998.

[Costa y Hertz, 1997] D. Costa y A. Hertz. Ants can color graphs, *J. of Operations Research Society* **48**, 1997: 295-305.

[Crama y Spieksma, 1992] Y. Crama & F. Spieksma. Approximation algorithms for the three-dimensional assignment problem with triangle inequalities, *European J. of Oper. Res.* **60**, 1992: 273-279.

[Dakin, 1965] R. J. Dakin. A Tree Search Algorithm for Mixed Integer Programming Problems, *Computer Journal* **8**, 1965: 250-255.

[Díaz et al., 1996] A. Díaz, F. Glover, H. Ghaziri, J. González, M. Laguna, P. Moscato y F. Tseng. *Optimization heurística y redes neuronales*, Editorial Paraninfo, S.A., Madrid-España, 1996.

[Dorigo y Gambardella, 1997] M. Dorigo & L. M. Gambardella. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem, *IEEE Transactions on Evolutionary Computation* **1**(1), 1997: 55-66.

[Dorigo et al., 1991] M. Dorigo, V. Maniezzo & A. Colorni. Ant System: An Autocatalytic Optimization Process, Technical Report 91-016, Dipartimento di Electtronica e Informazione-Politecnico di Milano, Italia, 1991.

[Dorigo et al., 1992] M. Dorigo, V. Maniezzo & A. Colorni. An investigation of some properties of an Ant algorithm. Proceeding of the Parallel Problem solving from Nature Conference (PPSN 92), Bruselas-Bélgica, 1992.

[Eshelman et al., 1989] L. Eshelman, R. Caruana & D. Shaffer. Biases in the Crossover Landscape, en J. David Shaffer, editor, Proccedings of the thriad Conference of Genetic Algorithms and Their Applications, Morgan Kaufmann Publishers, San Mateo California 1989: 10-19.

[Fabregas, 1991] L. Fábregas. *Sistemas de Información. Desarrollo, Implementación y Mantenimiento. Ciclo de desarrollo de sistemas*, Editorial Miru, C.A., Caracas-Venezuela, 1991.

[Feo y Resende, 1995] Th. Feo & M. Resende. Greedy Randomized Adaptative Search Procedures, *J. of Global Optimization* **6**, 1995: 109-133.

[Frieze, 1983] A. Frieze. Complexity of a 3-dimensional Assignment Problem, *Euro J. of Oper. Res.* **138**, 1983: 161-164.

[Gambardella y Dorigo, 1995] L. M. Gambardella & M. Dorigo. Ant-Q: A Reinforcement Learning Approach to the Travelling Salesman Problem, en A. Prieditis & S. Russell, editores, Proccedings of twelfth International Conference on Machine Learning, Morgan Kaufmann Publishers, 1995: 252-260.

[Gendreau, 2003] M. Gendreau. *An Introduction to Tabu Search*, Handbook of Metaheuristics, editado por F. Glover y G. Kochemberger, Kluver Academics Publishers, London, 2003: 37-54.

[Gillett, 1976] B. E. Gillett. *Introduction to Operations Research. A Computer-Oriented. Algorithmic Approach*, McGraw-Hill, U.S.A, 1976.

[Glover, 1989] F. Glover. Tabu Search: Part I, *ORSA J. on Computing* **1**, 1989: 190-206.

[Glover, 1995] F. Glover. Tabu Thresholding: Improved Search by Nonmonotonic Trayectories, *ORSA J. on Computing* **7** (4), 1995: 426-442.

[Glover y Klingmann, 1988] F. Glover & D. Klingmann. Layering strategies for creating exploitable structure in linear and integer programs, *Mathematical Programming* **40**, 1988: 165-182.

[Glover et al., 2000] F. Glover, M. Laguna & R. Martí. Fundamentals of Scatter Search and Path Relinking, *Control and Cybernetics* **29**(3), 2000: 653-684.

[Goffin, 1980] J. Goffin. The relaxation method for solving systems of linear inequalities, *Mathematics of Oper. Res.* **5**(3), 1980: 388-414.

[Gonzalez y Centeno, 2001] J. González & M. Centeno. Desarrollo de un programa para resolver el problema de asignación 3-dimensional a través de un algoritmo genético, *SABER* **13**(2), 2001: 123-126.

[Greenhalgh y Marshall, 2000] D. Greenhalgh & S. Marshall. Convergence criteria for genetic algorithms, *SIAM J. Compt.* **30**(1), 2000: 269-282.

[Grundel et al., 2007] D. A. Grundel, P. A. Krokhmal, C. A. S. Oliveira & P. M. Pardalos. On the number of local minima for the multidimensional assignment problem, *J. Comb. Optim.* **13**, 2007: 1-18.

[Grundel et al., 2004] D. A. Grundel, C. A. S. Oliveira & P. M. Pardalos. Asymptotic Properties of Random Multidimensional Assignment Problem, *Journal of Optimization Theory and Applications* **122**(3), 2004: 487-500.

[Grundel y Pardalos, 2005] D. A. Grundel & P. M. Pardalos. Test Problem Generator for the Multidimensional Assignment Problem, *Computational Optimization and Applications* **30**, 2005: 133-146.

[Gue y Thomas, 1971] R. Gue & M. Thomas. *Mathematical Metods in Operations Research*, The MacMillan Company, Fourth Printing, New York - USA, 1971.

[Guignard, 2003] M. Guignard. Lagrangean Relaxation, *Sociedad de Estadística e Investigación de Operativa* **11**(2), 2003: 151-228.

[Gutin y Karapetyan, 2009a] G. Gutin & D. Karapetyan. Local search heuristics for the multidimensional assignment problem, *Lecture Notes in Computer Science* **5420**, 2009a: 100-115.

[Gutin y Karapetyan, 2009b] G. Gutin & D. Karapetyan. A memetic algorithm for the multidimensional assignment problem, *Lecture Notes in Computer Science* **5752**, 2009b: 125-129.

[Hansen y Jaumard, 1990] P. Hansen & B. Jaumard. Algorithms for the Maximun Satisfiability Problem, *Computing* **44**, 1990: 279-303.

[Held y Karp, 1971] M. Held & R. M. Karp. The travelling-salesman problem and minimun spanning trees: Part II, *Math. Prog.* **1**, 1971: 6-25.

[Held et al., 1974] M. Held, P. Wolfe & H. D. Crowder. Validation of subgradient optimization, *Math. Prog.* **6**, 1974: 62-88.

[Hu, 1982] T. C. Hu. *Combinatorial Algorithms*, Addison Wesley Publishing Company, Philippines, 1982.

[Huang y Lim, 2006] G. Huang & A. Lim. A hybrid genetic algorithm for the Three-Index Assignment Problem, *European Journal of Operational Research* **172**, 2006: 249-257.

[Karapetyan y Gutin, 2010] D. Karapetyan & G. Gutin. A new approach to population sizing for memetic algorithms: a case study for the

multidimensional assignment problem, to appear in *Evolutionary Computation* 2010.

[Kirpatrick et al., 1983] S. Kirkpatrick, C. Gelatt & P. Vecchi. Optimization by simulated annealing, *Science* **220**, 1983: 671-680.

[Krokhmal et al., 2007] P. A. Krokhmal, D. A. Grundel & P. M. Pardalos. Assymptotic behavior of the expected optimalvalue of the multidimensional assignment problem, *Math. Program. Ser. B* **109**, 2007: 525-551.

[Land y Doig, 1960] A. H. Land & A. G. Doig. AN AUtomatic Method for Olving Discrete Programming Problems, *Econometrica* **28**, 1960: 497-520.

[Levitin y Rubinovitz, 1993] G. Levitin & J. Rubinovitz. Genetic Algorithms for Linear and Cyclic Assignment Problem, *Computers Ops. Res.* **20**(6), 1993: 575-586.

[Magos, 1996] D. Magos. Tabu search for the planar three-index assignment problem, *J. of Global Optimization* **8**, 1996: 35-48.

[Magos y Miliotis, 1994] D. Magos & P. Miliotis. An algorithm for the planar three-index assignment problem, *European J. of Operational Res.* **77**, 1994: 141-153.

[Martí, 2002] R. Martí. Procedimientos Metaheurísticos en Optimización Combinatoria, Departamento d'Estadística i Investigació Operativa. Facultad de Matamàtiques. Universitat de València, 2002.

[Marval et al., 2004] F. Marval, M.V. Centeno & J.J. Salazar. Heurístico para balancear una línea de ensamblaje simple (SALBP, problema tipo 1) *Divulgaciones Matemáticas* **12**(1), 2004: 25-34.

[Metropolis et al., 1953] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller & E. Teller. Equation of State Calculations by Fast Computing Machines, *J. of Chemical Physics* **21**, 1953: 1087-1092.

[Moreno y Moreno, 1999] J. M. Moreno & J. A. Moreno. *Heurísticas en Optimización*, Dirección General de Universidades e Investigación. Consejería de Educ., Cult. y Deportes. Gobierno de Canarias, Tenerife-España, 1999.

[Moscato y Cotta, 2003] P. Moscato & C. Cotta. *A Gentle Introduction to Memetic Algoritms*, Handbook of Metaheuristics, editado por F. Glover y G. Kochemberger, Kluver Academics Publishers, London, 2003: 105-144.

[Nasberg et al., 1985] M. Näsberg, K. O. Jörnsten & P. A. Smeds. Variable Splitting - A new Lagrangean relaxation approach to some mathematical programming problems, Report LITH-MATH-R-85-04. Linkoping University, 1985.

[Nemhauser y Wolsey, 1998] G. Nemhauser & L. Wolsey. *Integer and Combinatorial Optimization*, John Wiley and Sons, 1998.

[Oliveira y Pardalos, 2004] C. A S. Oliveira & P. M. Pardalos. Randomized parallel algorithms for the multidimensional assignment problem, *Applied Numerical Mathematics* **49**, 2004: 117-133.

[Pasiliao, 2003] E. L. Pasiliao. Algorithms for Multidimensional Assignment Problems, Ph. D. thesis, Department of Industrial and Systems Engineering, University of Florida, 2003.

[Pierskalla, 1968] W. Pierskalla. The Multidimensional Assignment Problem, *Oper. Res.* **16**, 1968: 422-431.

[Pressman, 2002] R. Pressman. *Ingeniería de Software. Un enfoque práctico*, Quinta edición., Editorial MacGraw-Hill, Madrid-España, 2002.

[Poore y Robertson, 1997] A. B. Poore & A. J. Robertson. A New Lagrangian Relaxation Based Algorithm for a Class of Multidimensional Assignment Problems, *Computational Optimization and Applications* **8**, 1997: 129-150.

[Qi y Sun, 1999] L. Qi & D. Sun. *Polyhedral methods for solving three index assignment problem*, Supported by the Australian Research Council, <http://www.maths.unsw.edu.au/applied/report/>, 1999.

[Raghavarao, 1971] D. Raghavarao. *Constructions and Combinatorial Problems in Design of Experiments*, John Wiley and Sons, USA, 1971.

[Rayser, 1963] H. Rayser. *Combinatorial Mathematics*, The Mathematical Association of America, USA, 1963.

[Reeves, 1995] C. Reeves. *Modern Heuristic Techniques for Combinatorial Problems*, McGraw-Hill, UK. 1995.

[Reeves, 2003] C. Reeves. *Genetic Algoritms*, Handbook of Metaheuristics, editado por F. Glover y G. Kochemberger, Kluver Academics Publishers, London, 2003: 55-81.

[Ribeiro y Minoux, 1996] C. Ribeiro & M. Minoux. Solving hard constrained shortest path problems by Lagrangean relaxation an branch-and-bound algorithms. *Math. of Oper. Res.* **53**, 1986: 303-316.

[Robertson, 2001] A. J. Robertson. A Set of Greedy Randomized Adaptative Local Search Procedure (GRASP) Implementations for the Multidimensional Assignment Problem, *Computational Optimization and Applications* **19**, 2001: 145-164.

[Salazar, 2001] J. J. Salazar. *Programación Matemática*, Ediciones Díaz de Santos, S.A., Madrid-España, 2001.

[Shepardson y Marsten, 1980] F. Shepardson & R. E. Marsten. A Lagrangean relaxation algorithm for two-duty scheduling problem. *Management Science* **26**, 1980: 274-281.

[Sloane, 2003] N. Sloane. *Sequences A002860/M2051, A000315/M3690 y A072377* , in The On-Line Encyclopedia of Integer Sequence, <http://www.research.att.com/ njas/sequences/>. 2003.

[Soenen, 1977] R. Soenen. Contribution *à* l'étude des systèmes de conduite en temps réel en vue de la commande d'unités de fabrication. Thèse de Doctorat d'Etat, Université de Lille, France. 1977.

[Syswerda, 1989] G. Syswerda. Uniform Crossver in Genetic Algorithms, en J. David Shaffer, editor, *Proccedings of the thriad Conference of Genetic Algorithms and Their Applications, Morgan Kaufmann Publishers*, San Mateo California, 1989: 2-9.

[Taha, 1991] H. Taha. *Investigación de Operaciones*. Segunda Edición. Ediciones Alfaomega, S.A de C.V. México. D.F. 1991.

[Trudeau, 1993] R. Trudeau. *Introduction to Graph Theory*, Dover Publications, Inc., Toronto-USA, 1993.

[Vignaux y Michalewicz, 1991] G. Vignaux & Z. Michalewicz. A genetic algorithm for the transpotation problem, *IEEE Transaction System Man Cybernetic* **21**(2), 1991: 445-452.

[Vlach, 1967] M. Vlach. Branch and Bound method for the three-index assignment problem, *Ekonomick-Matematicky Obsor.* **3**, 1967: 181-191.

[Weissteins, 2003] E. Weissteins. *Latin Square*, <http://mathworld.wolfram.com>, 2003.

[Wilson, 1983] R. Wilson. *Introducción a la teoría de grafos*, Alianza Editorial, Madrid-España, 1983.

[Wolsey, 1998] L. Wolsey. *Integer Programming*, John Wiley & Sons, New York-USA, 1998.