VNIVERSIDAD NACIONAL AVFNOMA DE MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA

FACULTAD DE INGENIERÍA

UNA APROXIMACIÓN A LA LOCALIZACIÓN ÓPTIMA DE LOS LABORATORIOS DEL CIA

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRA EN INGENIERÍA

INGENIERÍA DE SISTEMAS - INVESTIGACIÓN DE OPERACIONES

P R E S E N T A :

MARÍA DEL CARMEN FERNÁNDEZ GARCÍA

TUTOR:
DR. MIGUEL ÁNGEL GUTIÉRREZ ANDRADE

2011

JURADO ASIGNADO:

Presidente: Dra. Idalia Flores de la Mota

Secretario: Dr. Javier Ramírez Rodríguez

Vocal: Dr. Miguel Ángel Gutiérrez Andrade

1er. Suplente: Dr. José de Jesús Acosta Flores

2do. Suplente: M. I. José Antonio Rivera Colmenero

Lugar o lugares donde se realizó la tesis:

Ciudad Universitaria, Facultad de Ingeniería, Universidad Nacional Autónoma de México

TUTOR DE TESIS:

Dr. Miguel Ángel Gutiérrez Andrade

A Javier y
a Héctor

Agradecimientos

Al Dr. Miguel Ángel Gutiérrez Andrade por: el tema de tesis, el tiempo que me dedicó, su ayuda, la claridad de sus ideas y la libertad que me dio.

A Roberto Vargas Querea por contarme del proyecto del Centro de Ingeniería Avanzada e invitarme a solicitar información junto con él.

En la División de Ingeniería Mecánica e Industrial al Dr. Leopoldo Adrián González González, al Dr. Jesús Manuel Dorador González, a la Ing. Miriam Graciela Mendoza Cano, a la Ing. Livier Baez Rivas, a la Ing. Arianna Ivett Sánchez Gutiérrez y a todas las personas involucradas en el proyecto de construcción del Centro de Ingeniería Avanzada que me proporcionaron información y me atendieron, por la prontitud con que me fue proporcionada dicha información, por su disposición para el trabajo y por su amabilidad, son un equipo excelente de nuestra Universidad Nacional Autónoma de México.

Al CONACYT por el año de beca que me otorgó.

Índice general

Resumen

El problema de asignación cuadrática es uno de los más complejos en la clase NP-completa, existen muchas aplicaciones al mismo, sin embargo, la disposición de instalaciones es la más popular. En esta tesis se discuten los métodos de solución tanto exactos como heurísticos, incluidas las estrategias metaheurísticas que se han empleado para resolverlo. Se resuelve el problema de asignación cuadrática sin restricciones mediante un algoritmo de búsqueda dispersa. Asimismo, se implementa un algoritmo de búsqueda dispersa para resolver el problema de asignación cuadrática con restricciones del Centro de Ingeniería Avanzada. Ambos algoritmos se basan en el algoritmo propuesto por Cung et al. en 1997. Y se reporta la experiencia computacional del algoritmo sin restricciones.

Introducción

Los problemas de optimización se dividen de manera natural en dos categorías: problemas con variables continuas y problemas con variables discretas. Los últimos se denominan problemas de optimización combinatoria, algunos ejemplos de esta clase de problemas son los siguientes:

Ejemplo 1. El problema del agente viajero. Un agente de ventas tiene que visitar n ciudades, comenzar y terminar en la misma ciudad y visitar sólo una vez cada ciudad. El desea encontrar la ruta que minimice la distancia recorrida. Esto es, sea $N = \{1, 2, \ldots, n\}$ un conjunto de n ciudades y $d(i, j) \in \mathbb{Z}^+$ la distancia entre las ciudades i, $j \in N$ se desea encontrar una permutación[1] $(\pi(1), \pi(2), ..., \pi(n))$ de $\{1, 2, ..., n\}$ que minimice

$$\left(\sum_{i=1}^{n-1} d(\pi(i), \pi(i+1)) \right) + d(\pi(n), \pi(1))$$

Ejemplo 2. El problema de asignación. Se tienen n trabajos y n personas para realizarlos Cada trabajo debe ser hecho por una persona y una persona puede hacer a lo más un trabajo. El costo de asignar el trabajo i a la persona j es c_{ij}, $x_{ij} = 1$ si el trabajo i se asigna a la persona j, $x_{ij} = 0$ en otro caso. Se desea encontrar una asignación de las personas a los trabajos que minimice el costo total de completar todos los trabajos.

$$\text{Min} \quad z = \sum_{j=1}^{n} \sum_{i=1}^{n} c_{ij} x_{ij}$$

[1] Una permutación $\pi = (\pi(1), \pi(2), ..., \pi(n))$ de un conjunto $N = \{1, 2, ..., n\}$ es un arreglo ordenado de los elementos del conjunto, donde $\pi(i) \in N$ representa el valor del elemento en la posición i del arreglo.

$$\text{sujeto a:} \quad \sum_{j=1}^{n} x_{ij} = 1 \text{ para } i = 1, ..., n$$
$$\sum_{i=1}^{n} x_{ij} = 1 \text{ para } j = 1, ..., n$$

El interés de la optimización combinatoria es el de desarrollar algoritmos que encuentren la solución en un tiempo razonable o como se dice en términos formales, en tiempo polinómico[2] a problemas como los dos antes mencionados.

El problema de asignación cuadrática (QAP, Quadratic Assignment Problem) es un problema de optimización combinatoria, que puede describirse como sigue: se tienen n terrenos y se van a construir n edificios, sean a_{ik} la distancia entre los terrenos i, k y b_{jl} el número de personas que se mueve entre los edificios j, l. Se desea determinar en qué terreno debe construirse cada edificio de manera que la distancia total recorrida por las personas sea mínima. Este planteamiento es aplicable a otros problemas tales como: el diseño de terminales en aeropuertos, en donde se quiere que los pasajeros que deban hacer un trasbordo recorran la distancia mínima entre una y otra terminal teniendo en cuenta el flujo de personas entre ellas; el diseño de plantas industriales, pues hay flujos continuos de personas entre las distintas partes de la planta y se requiere que la disposición de esta sea tal que la distancia que recorren los empleados para realizar sus labores sea mínima, etc.

Hay algoritmos que resuelven el problema de asignación cuadrática en forma exacta en menos de una hora cuando el número de terrenos y edificios a construir es menor a 13 (ver [GUT91]), sin embargo, este problema es NP-completo, como lo demostraron Sahni y González en 1976 ([SAH76]), lo que significa que no se conoce un algoritmo que encuentre en tiempo polinómico una solución óptima cuando la instancia[3] del problema es grande, por ejemplo, en la referencia [GON2000] el tiempo promedio de ejecución de la Unidad de Procesamiento Central que tomó encontrar una solución cercana a la mejor conocida para un problema particular de tamaño $n = 81$ fue de poco más que 1.28 días.

[2] En el Capítulo 1 se presenta la definición de algoritmo de tiempo polinomial.

[3] En el Capítulo 1 se da la definición de instancia y se hacen varios comentarios respecto este concepto.

El problema de asignación cuadrática es objeto de estudio de muchos investigadores porque ha sido trabajado con muchas aproximaciones, lo que lo hace un muy buen punto de comparación de técnicas y aproximaciones.

Este trabajo presenta el estado del arte del problema de asignación cuadrática. Como aportaciones del mismo se pueden citar las siguientes:

- Implementación de un algoritmo de búsqueda dispersa para el problema de asignación cuadrática.
- Se presenta la experiencia computacional de este trabajo de tesis.
- Se da solución a un problema real de asignación cuadrática, para el cual se implementó un algoritmo con restricciones.

En el Capítulo 1 se discuten la complejidad computacional de los algoritmos y una clasificación de las heurísticas. El Capítulo 2 trata el problema de asignación cuadrática y su estado del arte. En el Capítulo 3 se describen dos algoritmos de búsqueda dispersa para el problema de asignación cuadrática uno para el problema sin restricciones y otro para el problema con restricciones de espacio ambos toman como base el algoritmo que publicaron Cung et al. en 1997 (ver [CUN97]). Además se reporta la experiencia computacional del algoritmo sin restricciones. Hay dos apéndices. El Apéndice A contiene el programa del algoritmo sin restricciones y el Apéndice B el del algoritmo con restricciones. En el Capítulo 4 se soluciona un problema de asignación cuadrática real, encontrar la localización de los espacios dentro del edificio del Centro de Ingeniería Avanzada (CIA) que minimice las distancias que se recorren dentro del mismo.

Capítulo 1

Complejidad y heurísticas

En este capítulo se presenta la complejidad computacional y una clasificación de las heurísticas. La primera permite identificar la problemática del problema de asignación cuadrática, mientras que la segunda sirve para ubicar el algoritmo que se empleará para encontrar soluciones aproximadas a dicho problema. Para introducir estos temas se definirán varios conceptos fundamentales.

Comienzo con la noción de algoritmo. Un **algoritmo** es un procedimiento para resolver un problema que recibe datos de entrada, sigue una secuencia de instrucciones, cada una de las cuales puede estar compuesta por pasos elementales y produce una salida. Se puede pensar en él como en un programa de computadora escrito en un lenguaje concreto. Son ejemplos de pasos elementales: la asignación de una variable, el acceso a una variable cuyo índice está almacenado en otra variable, las operaciones aritméticas simples como la suma, resta, multiplicación, división, comparación de números, etc.

Al dar valores específicos a los datos de entrada de un problema se obtiene una **instancia**. Por ejemplo, una instancia del problema del agente viajero, que se planteó en el Ejemplo 1, es la dada por $N = \{1, 2, 3, 4\}$, $d(1, 2) = 9$, $d(1, 3) = 3$, $d(1, 4) = 2$, $d(2, 3) = 6$, $d(2, 4) = 4$, $d(3, 4) = 5$, donde $d(i, j) = d(j, i)$ $\forall i \neq j$; $i, j \in N$. La permutación $(1, 3, 2, 4)$ es una solución de esta instancia, pues corresponde a una ruta con la menor longitud posible, 15 unidades, ver figura 1.1.

El tiempo requerido por un algoritmo se expresará en términos de una sola variable, el "tamaño" de una instancia del problema, pues se espera que la dificultad de una instancia del problema varíe con su tamaño. El tamaño de una instancia de un problema frecuentemente se mide de manera informal.

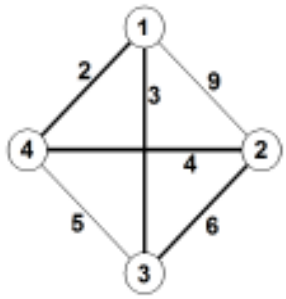


Figura 1.1: Una instancia del problema del agente viajero y una ruta con la menor longitud posible, 15 unidades.

Por ejemplo, para el problema del agente viajero se usa como tamaño de la instancia a n, el número de ciudades. No obstante, una instancia del problema con n ciudades incluye además de las etiquetas de las n ciudades a las $n(n-1)/2$ distancias entre las ciudades, es decir, estos $n(n-1)/2$ datos también forman parte de los datos de entrada.

La computadora recibe como entrada una instancia del problema, la cual puede describirse con una cadena finita de símbolos de un alfabeto finito con un esquema de codificación fijo. La longitud de esta cadena es usada como la medida formal del tamaño de la instancia.

Por ejemplo, las instancias del problema del agente viajero pueden describirse usando el alfabeto $\{-, /, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Si el ejemplo previo es codificado con este alfabeto por la cadena "1-2-3-4//9/3/2//6/4//5", la longitud de la entrada del ejemplo es 21.

Una **codificación** de un conjunto X de objetos abstractos es un mapeo m de X en un conjunto de cadenas binarias[1]. Por ejemplo, la codificación de los números naturales $\mathbb{N} = \{0, 1, 2, 3, 4, ...\}$ en cadenas binarias es {0, 1, 10, 11, 100, ...}. Usando esta codificación $m(25) = 11001$. Así la longitud de $x = 25$ es 5. Cualquier objeto puede ser codificado con una cadena binaria.

La complejidad de tiempo de un algoritmo expresa sus requerimientos de tiempo dando para cada posible longitud de entrada, una medida del tiempo requerido en el peor de los casos, para que el algoritmo resuelva una instancia de un problema de ese tamaño.

[1]No es necesario que las cadenas sean binarias, basta con que cada elemento de la cadena pertenezca a un conjunto finito con al menos dos elementos.

1.1. Tiempo de corrida de un algoritmo

Sea A un algoritmo cuyos datos de entrada son elementos de un conjunto X y sea $p : X \to \mathbb{R}^+$. Si existe una constante $\alpha > 0$ tal que A termine sus cálculos después de a lo más $\alpha p(x)$ pasos elementales para cada $x \in X$, se dice que A **corre en tiempo** $O(p)$**.** También se dice que el **tiempo de corrida** (o **complejidad de tiempo**) **de** A es $O(p)$. Si p es un polinomio, se dice que A es un **algoritmo de tiempo polinomial**.

Suponga que el conjunto de instancias de un problema es un conjunto de cadenas binarias. Un algoritmo resuelve una instancia i de un problema en tiempo $O(p(n))$ si cuando la longitud de la cadena binaria de la instancia i es n, el algoritmo puede producir la solución en un tiempo $O(p(n))$.

La siguiente tabla muestra los tiempos de corrida de algunos algoritmos con complejidades de tiempo distintas. Para diferentes longitudes de entrada n se muestra el tiempo de corrida de algoritmos que realizan n^2-n, $6n^3+12n$, n^4, 2^n y $n!$ pasos elementales, bajo el supuesto de que un paso elemental toma un nanosegundo en efectuarse.

n	$n^2 - n$	$6n^3 + 12n$	n^4	2^n	$n!$
10	90 ns	6 μs	10 μs	1 μs	4 ms
15	210 ns	20 μs	51 μs	33 μs	22 min
20	380 ns	48 μs	160 μs	1 ms	77 años
40	2 μs	384 μs	3 ms	18 min	
60	4 μs	1 ms	13 ms	37 años	
80	6 μs	3 ms	41 ms	$4 \cdot 10^7$ años	
100	10 μs	6 ms	100 ms		
10^3	1 ms	6 s	17 min		
10^4	100 ms	2 h	116 días		
10^5	10 s	69 días	3171 años		
10^6	17 min	190 años			
10^7	1 día				
10^8	116 días				

Como se observa en la tabla, los algoritmos de tiempo polinomial son más rápidos para instancias suficientemente grandes. Por lo anterior, a veces estos algoritmos son llamados buenos o eficientes.

1.2. Las clases P, NP y NP-completo

Se vio lo que es un algoritmo de tiempo polinomial, resulta natural preguntarse si todos los problemas pueden resolverse por algoritmos de tiempo polinomial. La respuesta es no. Hay problemas para los que se sabe que no existen algoritmos de tiempo polinomial, problemas para los que no existen algoritmos, (por ejemplo, un problema que puede ser resuelto en un tiempo finito pero no en tiempo polinomial es decidir si una expresión regular extendida denota al conjunto vacío, ver [AHO74, p. 422]. Un problema para el que no existe un algoritmo que lo resuelva es el "problema de la detención" de Turing.) problemas NP y problemas $NP - completos$, no se han descubierto algoritmos de tiempo polinomial para estos últimos y nadie ha podido probar todavía que no pueden existir algoritmos de tiempo polinomial para ellos, desde que se planteó en 1971 si un problema $NP - completo$ podía ser resuelto por un algoritmo de tiempo polinomial.

La clase $NP - completo$ tiene que ver con problemas de decisión, en los cuales la respuesta es "sí" o "no" (o bien, "1" o "0"). Aunque mostrar que un problema es $NP - completo$ me restringe al dominio de los problemas de decisión, usualmente se puede modelar un problema de optimización dado con un problema de decisión relacionado imponiendo un límite al valor a ser optimizado.

Uno de los convenientes de enfocarse en problemas de decisión es que facilitan el uso de la maquinaria de la teoría de lenguajes formales. Vale la pena revisar algunas definiciones de esta teoría. Un **alfabeto** $\aleph$ es un conjunto finito de símbolos con al menos dos elementos. Un **lenguaje** L sobre $\aleph$ es cualquier conjunto de cadenas compuestas de símbolos de $\aleph$. Por ejemplo, si $\aleph = \{0, 1\}$ el conjunto $L = \{10, 11, 101, 111, 1011, 1101, 10001, ...\}$ es el lenguaje de representaciones binarias de números primos. Se denotará la **cadena vacía** por ε, y al **lenguaje vacío** por $\varnothing$. El lenguaje de todas las cadenas sobre $\aleph$ se denota $\aleph^*$. Por ejemplo, si $\aleph = \{0, 1\}$ entonces $\aleph^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, ...\}$ es el conjunto de todas las cadenas binarias. Cada lenguaje L sobre $\aleph$ es un subconjunto de $\aleph^*$.

Desde el punto de vista de la teoría de lenguajes, el conjunto de instancias de cualquier problema de decisión Φ es el conjunto $\aleph^*$, donde $\aleph = \{0, 1\}$. Φ está caracterizado por aquellas instancias del problema que producen un 1 (sí) como respuesta, se puede ver a Φ como un lenguaje L sobre $\aleph = \{0, 1\}$, donde $L = \{x \in \aleph^* : \Phi(x) = 1\}$.

Se dice que un algoritmo A **acepta** una cadena $x \in \{0, 1\}^*$ si, al darle

como entrada x, la salida del algoritmo es $A(x) = 1$. El lenguaje **aceptado** por un algoritmo A es el conjunto de cadenas $L = \{x \in \{0,1\}^* : A(x) = 1\}$, esto es, el conjunto de cadenas que el algoritmo acepta. Un algoritmo **rechaza** una cadena x si $A(x) = 0$.

Aun si un lenguaje L es aceptado por un algoritmo A, el algoritmo no necesariamente rechaza una cadena $x \notin L$ que se le proporciona como entrada. Por ejemplo, el algoritmo se puede quedar atorado para siempre en esa cadena. Un lenguaje es **decidido** por un algoritmo A si cada cadena binaria en L es aceptada por A y cada cadena binaria que no está en L es rechazada por A. Un lenguaje L es **aceptado en tiempo polinomial** (**decidido en tiempo polinomial**) por un algoritmo A si es aceptado (decidido) por A y A es un algoritmo de tiempo polinomial. Así, para aceptar un lenguaje, un algoritmo sólo necesita preocuparse de las cadenas en L, pero para decidir un lenguaje, debe aceptar o rechazar correctamente cada cadena en $\{0,1\}^*$. Para el problema de detención de Turing existe un algoritmo de aceptación, pero no existe uno de decisión.

Usando la teoría de lenguajes se puede definir a la clase P.

1.2.1. La clase P

Sea Φ un problema de decisión y L el lenguaje que le corresponde, si L es decidido en tiempo polinomial por un algoritmo entonces Φ está en la clase P. Es decir, un problema de decisión está en la clase P si el lenguaje que le corresponde es decidido en tiempo polinomial por un algoritmo.

P es también la clase de problemas de decisión cuyos lenguajes correspondientes pueden ser aceptados en tiempo polinomial, para una demostración de esta afirmación ver [COR2001, pp. 977 y 978].

Para probar que un problema está en P uno usualmente describe un algoritmo de tiempo polinomial.

Generalmente estos problemas se consideran tratables, dos razones para ello son las siguientes:

Primera, un problema que puede ser resuelto en tiempo polinomial en un modelo de máquina, puede ser resuelto en tiempo polinomial en otro modelo. Por ejemplo, la clase de problemas que se pueden resolver en tiempo polinomial por una máquina de acceso aleatorio es la misma que la clase de problemas que se pueden resolver en tiempo polinomial por una máquina de Turing[2].

[2]Puede ver la definición de máquina de Turing en [KOR2000, p.328].

Es también la misma que la clase de problemas resolubles en tiempo polinomial en una computadora en paralelo cuando el número de procesadores crece polinomialmente con el tamaño de la entrada. Cada algoritmo puede, teóricamente, ser escrito como una máquina de Turing, con una pérdida en eficiencia que esta polinomialmente acotada, y cualquier algoritmo de tiempo polinomial es calculable en tiempo polinomial por una máquina de Turing [KOR2000].

Segunda, los problemas que se pueden resolver en tiempo polinomial tienen propiedades de cerradura, pues los polinomios son cerrados bajo la suma, multiplicación y composición. Por ejemplo, si la salida de un algoritmo de tiempo polinomial es la entrada de otro, el algoritmo compuesto es polinomial.

1.2.2. La clase NP

Se requiere que para las instancias del problema que producen un uno como respuesta haya un certificado el cual pueda ser checado en tiempo polinomial. Por ejemplo, en el problema del circuito hamiltoniano[3], dada una gráfica dirigida $G = (V, E)$, un certificado sería una secuencia $\langle v_1, v_2, v_3, ..., v_{|V|} \rangle$ de $|V|$ vértices, ya que se puede checar en tiempo polinomial que $(v_i, v_{i+1}) \in E$ para $i = 1, 2, 3, ..., |V| - 1$ y que $(v_{|V|}, v_1) \in E$ también.

Se define un **algoritmo de verificación** como un algoritmo A con dos entradas: una de ellas es una cadena de entrada ordinaria x y la otra es una cadena binaria y llamada **certificado**. Un algoritmo A con dos entradas **verifica** una cadena de entrada x si existe un certificado y tal que $A(x, y) = 1$. Un **lenguaje verificado** por un algoritmo de verificación A es

$$L = \{x \in \{0,1\}^* : \text{existe un certificado } y \in \{1,0\}^* \text{ tal que } A(x, y) = 1\}$$

Intuitivamente, un algoritmo A verifica un lenguaje L si para cualquier cadena $x \in L$, hay un certificado y que A puede usar para probar que $x \in L$. Más aún, para cualquier cadena $x \notin L$, no debe haber un certificado que pruebe que $x \in L$.

[3]Para una gráfica $G = (V, E)$ con V un conjunto de vértices y E un conjunto de aristas, un circuito simple en G es una secuencia $\langle v_1, v_2, ..., v_k \rangle$ de vértices distintos de V tales que $(v_i, v_{i+1}) \in E$ para $1 \leq i < k$ y $(v_k, v_1) \in E$. Un circuito hamiltoniano en G es un circuito simple que incluye todos los vértices de G. El problema del circuito hamiltoniano es el siguiente: dada una gráfica $G = (V, E)$ ¿G contiene un circuito hamiltoniano?

La clase NP es la clase de lenguajes que pueden ser verificados por un algoritmo de tiempo polinomial[4]. Más precisamente, un lenguaje L pertenece a NP si y sólo si existen un algoritmo A de dos entradas de tiempo polinomial y una constante c tales que

$$L = \{x \in \{0,1\}^* : \text{existe un certificado } y \text{ con } |y| = O(|x|^c) \text{ tal que } A(x,y) = 1\}$$

Se dice que el algoritmo A **verifica** al lenguaje L **en tiempo polinomial.**

Cualquier problema en P está en NP, esto es, si $L \in P$, entonces $L \in NP$, ya que si hay un algoritmo de tiempo polinomial que decide L, el algoritmo puede fácilmente convertirse en un algoritmo de verificación con dos entradas que simplemente ignore cualquier certificado y acepte exactamente las cadenas de entrada que determine están en L. Así $P \subseteq NP$. Es una pregunta abierta si P es o no un subconjunto propio de NP, es decir, es una pregunta abierta si $P = NP$.

Muchos científicos de teoría de la computación creen que P y NP no son la misma clase, que NP incluye lenguajes que no están en P.

1.2.3. La clase NP-completo

Quizás la razón que obliga a los científicos de teoría de la computación a creer que $P \neq NP$ es la existencia de la clase de problemas $NP - completo$. Esta clase tiene la sorprendente propiedad de que si algún problema $NP - completo$ puede ser resuelto en tiempo polinomial, entonces cada problema en NP tiene una solución en tiempo polinomial, esto es, $P = NP$. Sin embargo, a pesar de años de estudio y del amplio rango de problemas $NP-completos$ que han sido estudiados, nadie ha descubierto una solución de tiempo polinomial para alguno de ellos, sería en verdad asombroso si todos ellos pudieran ser resueltos en tiempo polinomial.

Los lenguajes $NP - completos$ son, en un sentido, los lenguajes "más difíciles" en NP. En esta sección se verá como comparar la "dureza" relativa de los lenguajes usando una noción llamada "reductibilidad en tiempo polinomial".

[4]El nombre NP significa "no-determinista polinomial". La clase NP fue estudiada originalmente en el contexto del no determinismo, se usará una noción equivalente de verificación, la cual es un poco más simple todavía. En [KOR2000] se presentan las clases NP y $NP - completo$ en términos de algoritmos no deterministas.

Reductibilidad

Intuitivamente, un problema Φ puede ser reducido a otro problema Φ' si cualquier instancia de Φ puede ser "fácilmente reexpresada" como una instancia de Φ', la solución de la cual proveerá una solución a la instancia de Φ. Por ejemplo, el problema de resolver ecuaciones lineales se reduce al problema de resolver ecuaciones cuadráticas. Dada una instancia $ax + b = 0$, ésta se transforma en $0x^2 + ax + b = 0$, cuya solución provee una solución a $ax + b = 0$. Así, si un problema Φ se reduce a otro problema Φ', entonces Φ es, en un sentido "no más difícil de resolver" que Φ'.

Regresando a los lenguajes formales para problemas de decisión, se dice que un lenguaje L_1 es **reducible en tiempo polinomial** a un lenguaje L_2, y se denota por, $L_1 \leq_P L_2$, si existe una función calculable en tiempo polinomial $f : \{0,1\}^* \to \{0,1\}^*$ tal que para todo $x \in \{0,1\}^*$

$$x \in L_1 \text{ si y sólo si } f(x) \in L_2$$

se llama a la función f **función de reducción**, el algoritmo de tiempo polinomial F que calcula f es llamado **algoritmo de reducción.**

Cada lenguaje es un subconjunto de $\{0,1\}^*$. La función de reducción f provee un mapeo de tiempo polinomial tal que si $x \in L_1$, entonces $f(x) \in L_2$. Más aún, si $x \notin L_1$, entonces $f(x) \notin L_2$. Así, la función de reducción mapea cualquier instancia x del problema de decisión representado por el lenguaje L_1 en una instancia $f(x)$ del problema representado por L_2. Proporcionando una respuesta si $f(x) \in L_2$ que directamente proporciona una respuesta si $x \in L_1$.

La reducción en tiempo polinomial da una poderosa herramienta para probar que varios lenguajes pertenecen a P.

Lema. Si $L_1, L_2 \subseteq \{0,1\}^*$ son lenguajes tales que $L_1 \leq_P L_2$, entonces $L_2 \in P$ implica $L_1 \in P$.

Demostración. Sea A_2 un algoritmo de tiempo polinomial que decide L_2, y sea F un algoritmo de reducción de tiempo polinomial que calcula la función de reducción f. Se construirá un algoritmo de tiempo polinomial A_1 que decide a L_1.

Sea $x \in \{0,1\}^*$, el algoritmo A_1 usa a F para transformar x en $f(x)$, luego aplica el algoritmo A_2 para determinar si $f(x) \in L_2$. La salida de A_2 es la salida de A_1, pues el lenguaje L_1 es reducible en tiempo polinomial al

lenguaje L_2. Por lo que A_1 corre en tiempo polinomial, ya que tanto F como A_2 corren en tiempo polinomial.

Un lenguaje $L \subseteq \{0,1\}^*$ es $\mathbf{NP-completo}$ si

1. $L \in NP$, y
2. $L' \leq_P L$ para cada $L' \in NP$

Si un lenguaje L satisface la propiedad 2, pero no necesariamente la propiedad 1, se dice que L es $\mathbf{NP-difícil}$.

Teorema. Si algún problema $NP-completo$ es resoluble en tiempo polinomial, entonces $P = NP$. Equivalentemente, si algún problema en NP no es resoluble en tiempo polinomial, entonces ningún problema $NP-completo$ es resoluble en tiempo polinomial.

Demostración. Suponga que $L \in P$ y que también L es $NP-completo$. Como para cada $L' \in NP$ se tiene que $L' \leq_P L$ por la propiedad 2 de los lenguajes $NP-completos$, y por el lema $L' \in P$, entonces $P = NP$.

La contrapositiva de la afirmación que se acaba de demostrar es:

Si $P \neq NP$ entonces ningún problema $NP-completo$ es resoluble en tiempo polinomial. Se sabe que $P \subseteq NP$, así que $P \neq NP$ porque $NP \nsubseteq P$, es decir, porque algún problema en NP no se puede resolver en tiempo polinomial, por lo que se llega a:

Si algún problema NP no se puede resolver en tiempo polinomial, entonces ningún problema $NP-completo$ es resoluble en tiempo polinomial, que es la segunda afirmación del teorema.

Es por esta razón que las investigaciones respecto a si $P \neq NP$ se centran alrededor de los problemas $NP-completos$. Se sabe que si alguien proporciona un algoritmo de tiempo polinomial para un problema $NP-completo$ probará que $P = NP$. Sin embargo, no se han descubierto algoritmos de tiempo polinomial para ningún problema $NP-completo$, lo cual provee excelente evidencia de la intratabilidad de los problemas $NP-completos$.

Si se afirma que un problema es $NP-completo$ se debe proveer evidencia de su intratabilidad. La referencia [GAR79] da una larga lista de problemas de optimización combinatoria que pertenecen a la clase $NP-completo$, actualizada hasta 1978. Y provee técnicas para probar si un problema dado pertenece a esta clase, al igual que [COR2001] y [KOR2000]. Es importante

saber a qué clase pertenece el problema que se desea resolver ya que si se sabe que el problema en cuestión está en la clase $NP-completo$, no debe buscarse su solución óptima, a menos que la instancia del problema sea pequeña.

1.3. Técnicas heurísticas

El término heurística se deriva del griego *heuriskein* que significa encontrar o descubrir [REE93, p. 5]. Sin embargo, lo que en Investigación de Operaciones es ahora casi universalmente llamado heurística podría ser descrito mejor como una técnica de búsqueda, que no puede garantizar encontrar algo. El primer registro del término en *International Abstracts in Operations Research* parece haber ocurrido en 1960 (en un artículo publicado en francés), pero parece que la connotación generalmente aceptada de este término se estableció a mediados de 1960.

Se define una **heurística** como una técnica que busca buenas soluciones (es decir, cercanas al óptimo) a un costo computacional razonable sin poder garantizar factibilidad u optimalidad, o aún en muchos casos, sin poder establecer que tan cerca del óptimo está una solución factible particular.

Se ha trabajado mucho en técnicas heurísticas para resolver problemas combinatorios (ver [ZAN89]). La causa del interés en las heurísticas parece doble: por un lado, el desarrollo del concepto de complejidad computacional ha proporcionado una base racional para explorar las técnicas heurísticas en lugar de perseguir la optimalidad; y por otro, han surgido nuevas técnicas de propósito general muy eficientes para resolver problemas de optimización combinatoria en tiempos de cómputo razonables.

Las heurísticas son usualmente bastante más flexibles y son capaces de enfrentar funciones objetivo y/o restricciones más complicadas (y más realistas) que los algoritmos exactos. Por ejemplo, para técnicas como el recocido simulado, la búsqueda tabú y los algoritmos genéticos, la función objetivo no necesita supuestos de linealidad, así es posible modelar problemas del mundo real con más precisión.

No obstante, el uso de técnicas heurísticas también presenta inconvenientes. Uno de ellos es que por lo general no es posible conocer la calidad de la solución, es decir, cuan cerca del óptimo esta la solución heurística que nos ofrecen. Así, para evaluar los resultados de una heurística podemos emplear: el cálculo de cotas inferiores y superiores para la solución óptima del problema, el desempeño empírico de los problemas de prueba, o la inferencia

estadística.

Por último, para adentrarme en lo que son las heurísticas daré una clasificación de éstas.

1.3.1. Heurísticas de construcción

Los algoritmos de construcción, también denominados **heurísticas de un sólo paso**, generan una solución por agregación de componentes individuales (nodos, arcos, variables) uno a la vez, hasta obtener una solución factible. Los algoritmos **glotones**, los cuales buscan maximizar mejorando a cada paso, abarcan una clase grande de heurísticas de construcción. Por ejemplo, para el problema de la mochila entero:

$$\begin{array}{lrcccccccc} \text{Max} & z & = & c_1x_1 & + & \ldots & + & c_nx_n & & \\ & \text{s. a.} & & a_1x_1 & + & \ldots & + & a_nx_n & \leq & b \\ & & & x_i & = & 0,1 & & i & = & 1,...n \end{array}$$

Considere el siguiente caso particular con $n = 6$

$$\begin{array}{lrcccccccccccc} \text{Max} & z & = & 10x_1 & + & 5x_2 & + & 5x_3 & + & 2x_4 & + & 8x_5 & + & 9x_6 & & \\ & \text{s. a.} & & 7x_1 & + & 8x_2 & + & 5x_3 & + & 8x_4 & + & 7x_5 & + & 6x_6 & \leq & 15 \\ & & & & & & & & & x_i & = & 0,1 & & i & = & 1,...,6 \end{array}$$

El algoritmo glotón para el problema de la mochila entero consiste en dar en cada paso el valor de 1 a aquella variable x_i que tiene la mayor relación *costo/volumen* : c_i/a_i (es decir, se prefiere los productos con valor de venta alto y volumen pequeño) hasta que no exista más recurso disponible (espacio en la mochila). Para este ejemplo el orden de prioridad de las variables es:

Producto	Relación (c_i/a_i)	Prioridad
x_1	1,4	2
x_2	0,6	5
x_3	1,0	4
x_4	0,3	6
x_5	1,1	3
x_6	1,5	1

La secuencia de trabajo del algoritmo sería:

$$\begin{array}{rcl} x_6 & = & 1,\ z = 9,\ \text{recurso disponible} = 9 \\ x_1 & = & 1,\ z = 19,\ \text{recurso disponible} = 2 \end{array}$$

El algoritmo termina aquí porque el recurso disponible es insuficiente para que otra variable pueda asumir el valor 1. Así, el algoritmo glotón encuentra la siguiente solución:

$$\begin{aligned} z &= 19 \\ x_1 &= x_6 = 1 \\ x_2 &= x_3 = x_4 = x_5 = 0 \end{aligned}$$

En este caso el algoritmo glotón o goloso (como lo llaman algunos autores) encontró la solución óptima global, sin embargo, para problemas grandes este algoritmo no siempre encuentra el óptimo global.

1.3.2. Heurísticas de mejoramiento

Las heurísticas de mejoramiento comienzan con una solución factible y sucesivamente la mejoran por una secuencia de cambios o fusiones en una búsqueda local. Generalmente una solución factible es mantenida a lo largo del procedimiento.

Todas las heurísticas de búsqueda local (por vecindades) caen dentro de esta categoría. Por ejemplo, para el problema de la mochila entero:

$$\begin{array}{lrcrcrcrcrcrc} \text{Max} & z & = & 9x_1 & + & 3x_2 & + & 5x_3 & + & x_4 & + & 10x_5 & \\ & \text{s. a.} & & 2x_1 & + & 19x_2 & + & 7x_3 & + & 4x_4 & + & x_5 & \leq & 20 \\ & & & & & & & x_i & = & 0,1 & & i & = & 1,...,5 \end{array}$$

Se puede definir una vecindad local como el conjunto de vectores de ceros y unos que difieren de una solución factible dada en a lo más dos componentes. La búsqueda procede revisando todos los vecinos de la solución factible dada y moviéndose al que tenga el mejor valor en la función objetivo, si existe. Si no se encuentra ninguno, entonces se obtiene un óptimo local. Sea $x = (1, 0, 0, 0, 1)$ la solución factible inicial.

Solución	Restricción	Valor
$(1, 0, 0, 0, 1)$	$3 \leq 20$	$z = 19$

Los vecinos de x son:

	Vecino	Restricción	Valor
1	$(0,0,0,0,1)$	$1 \leq 20$	$z = 10$
2	$(1,1,0,0,1)$	$22 \nleq 20$	Sol. infactible
3	$(1,0,1,0,1)$	$10 \leq 20$	$z = 24$
4	$(1,0,0,1,1)$	$7 \leq 20$	$z = 20$
5	$(1,0,0,0,0)$	$2 \leq 20$	$z = 9$
6	$(0,1,0,0,1)$	$20 \leq 20$	$z = 13$
7	$(0,0,1,0,1)$	$8 \leq 20$	$z = 15$
8	$(0,0,0,1,1)$	$5 \leq 20$	$z = 11$
9	$(0,0,0,0,0)$	$0 \leq 20$	$z = 0$
10	$(1,1,1,0,1)$	$29 \nleq 20$	Sol. infactible
11	$(1,1,0,1,1)$	$26 \nleq 20$	Sol. infactible
12	$(1,1,0,0,0)$	$21 \nleq 20$	Sol. infactible
13	$(1,0,1,1,1)$	$14 \leq 20$	$z = 25$
14	$(1,0,1,0,0)$	$9 \leq 20$	$z = 14$
15	$(1,0,0,1,0)$	$6 \leq 20$	$z = 10$

Como el vecino de x con el mejor valor de la función objetivo es $y = (1,0,1,1,1)$, me muevo a y, los vecinos de y son:

	Vecino		Vecino		Vecino
1	$(0,0,1,1,1)$	6	$(0,1,1,1,1)$	11	$(1,1,1,0,1)$
2	$(1,1,1,1,1)$	7	$(0,0,0,1,1)$	12	$(1,1,1,1,0)$
3	$(1,0,0,1,1)$	8	$(0,0,1,0,1)$	13	$(1,0,0,0,1)$
4	$(1,0,1,0,1)$	9	$(0,0,1,1,0)$	14	$(1,0,0,1,0)$
5	$(1,0,1,1,0)$	10	$(1,1,0,1,1)$	15	$(1,0,1,0,0)$

Los vecinos 3, 4, 7, 8, 10, 11, 14 y 15 de y son los vecinos 4, 3, 8, 7, 11, 10, 15 y 14 de x, respectivamente, que ya se habían revisado. El vecino 13 de y es x. Los vecinos de y que falta evaluar son los siguientes seis:

	Vecino	Restricción	Valor
1	$(0,0,1,1,1)$	$12 \leq 20$	$z = 16$
2	$(1,1,1,1,1)$	$33 \nleq 20$	Sol. infactible
5	$(1,0,1,1,0)$	$13 \leq 30$	$z = 15$
6	$(0,1,1,1,1)$	$31 \nleq 20$	Sol. infactible
9	$(0,0,1,1,0)$	$11 \leq 20$	$z = 6$
12	$(1,1,1,1,0)$	$32 \nleq 20$	Sol. infactible

El valor de la función objetivo en $u = (1, 0, 1, 1, 1)$ es $z = 25$, y ningún vecino de u mejora éste valor, así que u es un óptimo local. En este caso particular u es el óptimo global de este problema de la mochila entero.

1.3.3. Heurísticas de descomposición o división

Estos algoritmos dividen o descomponen el problema en varios problemas tratables más pequeños, la salida de uno es la entrada del siguiente, de forma que al resolverlos todos se obtenga una solución para el problema global. En esencia, el algoritmo de solución es descompuesto en un número discreto de pasos.

Se puede usar esta aproximación para resolver el problema del agente viajero. Considere la siguiente heurística de división. Se divide las n ciudades en m grupos, $m \geq 2$. Se determinan todas las formas (k) en que se puede dividir las n ciudades en m grupos. En cada grupo se determina un recorrido de peso mínimo. Se suma el peso de los recorridos de los m grupos. De las k formas en que se puede dividir las n ciudades en m grupos se elige la de menor peso. Finalmente se unen los m grupos con los arcos de menor peso, de esta manera se obtiene la ruta completa del agente viajero. Por ejemplo, considere la siguiente instancia del problema del agente viajero $N = \{1, 2, 3, 4, 5\}$, $d(1, 2) = 7$, $d(1, 3) = 8$, $d(1, 4) = 2$, $d(1, 5) = 4$, $d(2, 3) = 9$, $d(2, 4) = 7$, $d(2, 5) = 5$, $d(3, 4) = 2$, $d(3, 5) = 10$, $d(4, 5) = 2$, donde $d(i, j) = d(j, i)$ $\forall i \neq j$; $i, j \in N$, ver figura 1.2.

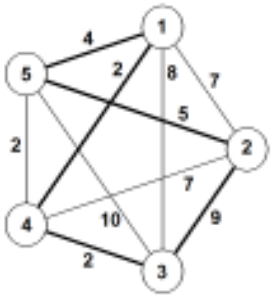


Figura 1.2: Una instancia del problema del agente viajero y una ruta de longitud 22, que es el mínimo posible en este caso.

Se dividen los elementos de N en dos conjuntos: uno con dos elementos y otro con los tres restantes. Las k formas en que se pueden dividir las 5

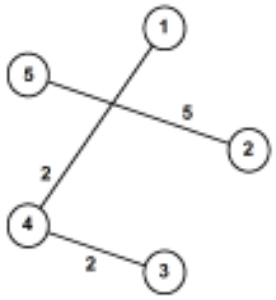


Figura 1.3: Arcos de menor peso que unen los nodos 2,5 y 1,3,4

ciudades en 2 grupos son las formas en que se pueden elegir dos de los cinco elementos de N, y son (las combinaciones de cinco en dos, $\binom{5}{2} = \frac{5!}{2!(5-2)!} = 10$) las siguientes. En cada caso: se elige entre los arcos que unen los tres nodos del último conjunto los dos de menor peso; y se suma el peso de los recorridos de los dos grupos.

Nodos	Peso del arco que los une	Nodos	Arcos que se eligieron para unirlos	Peso de los arcos elegidos
1, 2	7	3, 4, 5	(3,4), (4,5)	11
1, 3	8	2, 4, 5	(2,5), (4,5)	15
1, 4	2	2, 3, 5	(2,3), (2,5)	16
1, 5	4	2, 3, 4	(2,4), (3,4)	13
2, 3	9	1, 4, 5	(1,4), (1,5)	13
2, 4	7	1, 3, 5	(1,3), (1,5)	19
2, 5	5	1, 3, 4	(1,4), (3,4)	9
3, 4	2	1, 2, 5	(1,5),(2,5)	11
3, 5	10	1, 2, 4	(1,2),(1,4)	19
4, 5	2	1, 2, 3	(1,2),(1,3)	17

De las $k = 10$ formas en que se puede dividir las 5 ciudades en 2 grupos se elige la de menor peso, es decir, se eligen los arcos (2,5), (1,4), (3,4), ver figura 1.3.

Para tener una ruta que pase una sola vez por cada nodo, que empiece y termine en el mismo nodo los arcos que se pueden agregar son:

Arcos a agregar	Peso de los arcos
(1,5), (2,3)	13
(1,2), (3,5)	17

Finalmente se unen los dos grupos con los dos primeros arcos por ser los de menor peso. Así, se obtuvo como solución la permutación $(1, 4, 3, 2, 5)$, la ruta del agente viajero en esta permutación es de 22 unidades (ver figura 1.2). En este caso esta heurística de división encontró una solución óptima global, esto no siempre ocurre.

1.3.4. Heurísticas de relajación

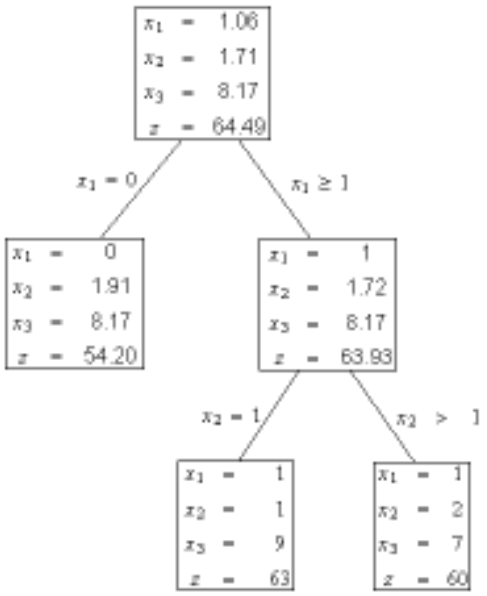
Esta categoría se refiere a expandir el espacio de soluciones para obtener un problema tratable. Estas técnicas son multietapas, ya que el resultado después de la relajación es infactible generalmente. Usualmente se requiere un segundo paso para obtener factibilidad.

Por ejemplo, para resolver por ramificación y acotamiento el problema de programación lineal entera

$$\begin{array}{lrcrcrcrl} \text{Max} & z & = & 11x_1 & + & 7x_2 & + & 5x_3 & \\ & \text{s. a.} & & 5x_1 & + & 0.9x_2 & + & x_3 & \leq 15 \\ & & & x_1 & + & 5.5x_2 & + & 3x_3 & \leq 35 \\ & & & 2x_1 & + & x_2 & + & x_3 & \leq 12 \\ & & & & & x_1, & x_2, & x_3 & \geq 0 \\ & & & & & x_1, & x_2, & x_3 & \in \mathbb{Z} \end{array}$$

La relajación consiste en ignorar la restricción de que las variables son enteras, es decir, se aplica el método simplex al problema de programación lineal sin la última restricción. La solución óptima que se obtiene es: x_1 =1.06, x_2 =1.71, x_3 =8.17, z =64.49. Ahora se va a considerar una a una las variables y a determinar los valores que pueden tomar. Se consideraron dos casos para x_1, $x_1 = 0$ o $x_1 \geq 1$. Si $x_1 = 0$, se deja fijo el valor de x_3 (x_3 =8.17) y se le da a x_2 el mayor valor que puede tomar cumpliendo las restricciones, x_2 =1.91, con estos valores el valor de la función objetivo es z =54.20. Para el caso $x_1 \geq 1$, se hace $x_1 = 1$, se deja fijo el valor de x_3 (x_3 =8.17) y se le da a x_2 el mayor valor que puede tomar cumpliendo las restricciones, x_2 =1.72, en este caso z =63.93. Como el valor de la función objetivo es mayor cuando $x_1 \geq 1$, hay que seguir por esta rama. Los casos que se consideraron ahora son: $x_2 = 1$ o $x_2 > 1$. En el caso $x_2 = 1$, se tenía $x_1 = 1$, por lo que se asigna a x_3 el mayor valor posible de manera que satisfaga las restricciones, $x_3 = 9$, con estos valores $z = 63$. Para el caso $x_2 > 1$, se hace $x_2 = 2$, se tenía $x_1 = 1$, y se le asigna a x_3 el mayor valor entero que puede tomar satisfaciendo las restricciones, esto es $x_3 = 7$ (no se le asigna el mayor valor posible $(7.\widehat{6})$

porque sólo se puede darle a x_3 el valor de 7, pues si se hace $x_3 = 8$ no se satisfacen las restricciones), con estos valores de las variables, $z = 60$. Así, la solución que se obtuvo para este problema es $x_1 = x_2 = 1$, $x_3 = 9$, $z = 63$. El siguiente diagrama resume el procedimiento.



Capítulo 2

El problema de asignación cuadrática

Una formulación matemática de este problema fue planteada inicialmente por Koopmans y Beckmann en 1957 ([KOO57]) y consiste en minimizar

$$f(\pi) = \sum_{i=1}^{n} \sum_{k=1}^{n} a_{ik} b_{\pi(i)\pi(k)}$$

donde π es una permutación del conjunto $N = \{1, \ldots, n\}$ y a_{ik}, $b_{\pi(i)\pi(k)}$ son números reales para $i, k = 1, \ldots, n$. Es decir, se desea encontrar una permutación π del conjunto N que minimice el valor de $f(\pi)$.

Para entender el problema suponga que existen n sitios disponibles y se van a construir n edificios en ellos. Sean a_{ik} la distancia entre los sitios i y k, y b_{jl} el número de personas que viajarán entre los edificios j y l. El problema es asignar los edificios a los sitios de construcción de manera que la distancia total recorrida sea mínima. Cada asignación corresponde a una permutación del conjunto $N = \{1, \ldots, n\}$, donde $\pi(i) = j$ significa que el edificio i es asignado al sitio j. Mientras que el producto $a_{ik} b_{\pi(i)\pi(k)}$ representa la distancia recorrida por las personas entre los edificios $j = \pi(i)$ y $l = \pi(k)$.

Ejemplo. Suponga que se va a construir un hospital y se desea localizar cuatro edificios A, B, C y D en cuatro sitios a, b, c y d, que las distancias en metros entre los cuatro edificios están dadas por la matriz de distancias A. El edificio A es de consultorios, el edificio B es la farmacia, el edificio C es para habitaciones de los internos y el edificio D son laboratorios. El flujo o movimiento entre estos edificios se describen por medio de la matriz de B,

donde b_{jl} es una medida para el número de personas y la frecuencia con la que viajan del edificio j al edificio l.

$$A = \begin{pmatrix} 0 & 15 & 40 & 30 \\ 15 & 0 & 25 & 20 \\ 40 & 25 & 0 & 35 \\ 30 & 20 & 35 & 0 \end{pmatrix} \qquad B = \begin{pmatrix} 0 & 17600 & 3200 & 1700 \\ 17600 & 0 & 80 & 850 \\ 3200 & 80 & 0 & 2400 \\ 1700 & 850 & 2400 & 0 \end{pmatrix}$$

Si el edificio A se construye en el sitio b, el edificio B en el sitio c, el edificio C en el sitio d y el edificio D en el sitio a, la permutación correspondiente a esta asignación es (2,3,4,1), es decir, $\pi(1) = 2, \pi(2) = 3, \pi(3) = 4$ y $\pi(4) = 1$. El valor de la función objetivo en esta permutación es el siguiente:

$$\begin{aligned} f(\pi) &= \sum_{i=1}^{4} (a_{i1}b_{\pi(i)\pi(1)} + a_{i2}b_{\pi(i)\pi(2)} + a_{i3}b_{\pi(i)\pi(3)} + a_{i4}b_{\pi(i)\pi(4)}) \\ &= \sum_{i=1}^{4} (a_{i1}b_{\pi(i)2} + a_{i2}b_{\pi(i)3} + a_{i3}b_{\pi(i)4} + a_{i4}b_{\pi(i)1}) \\ &= a_{11}b_{22} + a_{12}b_{23} + a_{13}b_{24} + a_{14}b_{21} + ... + a_{41}b_{12} + a_{42}b_{13} + a_{43}b_{14} + a_{44}b_{11} \\ &= 1,493,400 \end{aligned}$$

El espacio de soluciones de este problema es un conjunto que contiene 4!=24 permutaciones. En la tabla que se presenta a continuación se enlista cada permutación con su respectivo valor de la función objetivo.

Permutación	Valor	Permutación	Valor
π	$f(\pi)$	π	$f(\pi)$
(1,2,3,4)	1,092,000	(3,1,2,4)	1,253,900
(1,2,4,3)	1,069,700	(3,1,4,2)	1,141,300
(1,3,2,4)	1,765,500	(3,2,1,4)	1,435,400
(1,3,4,2)	1,470,700	(3,2,4,1)	1,251,900
(1,4,2,3)	1,795,100	(3,4,1,2)	1,683,800
(1,4,3,2)	1,522,600	(3,4,2,1)	1,612,900
(2,1,3,4)	981,400	(4,1,2,3)	1,276,600
(2,1,4,3)	981,800	(4,1,3,2)	1,163,600
(2,3,1,4)	1,836,400	(4,2,1,3)	1,412,700
(2,3,4,1)	1,493,400	(4,2,3,1)	1,251,500
(2,4,1,3)	1,843,300	(4,3,1,2)	1,654,200
(2,4,3,1)	1,499,900	(4,3,2,1)	1,606,000

En la tabla se observa que la permutación (2,1,3,4) es la solución para esta instancia ya que a esta asignación le corresponde el valor mínimo de la función objetivo, 981,400. Por lo anterior, el edificio A debe construirse en el sitio b, el B en el sitio a, el C en el sitio c y el edificio D en el d.

Existen muchas aplicaciones al problema de asignación cuadrática. Steinberg ([STE61]) lo usó para minimizar el número de conexiones entre los componentes en un pizarrón eléctrico; Pollatschek et al. ([POL76]) lo emplearon para definir el mejor diseño del teclado de una máquina de escribir y para paneles de control; Forsberg et al. ([FOR94]) lo usaron en el análisis de reacciones químicas; Rabak y Sichman ([RAB2003]), Miranda et al. ([MIR2005]) estudiaron la colocación de componentes electrónicos. Sin embargo, la disposición de instalaciones es la aplicación más popular del problema de asignación cuadrática: Dickey y Hopkins ([DIC72]) lo aplicaron para la asignación de edificios en un campus universitario, Elshafei ([ELS77]) en la planeación de un hospital.

Debido a su alta complejidad computacional, el QAP se eligió como la primera mejor prueba de aplicación para el proyecto GRIBB (GReat International Branch-and-Bound search). Este proyecto busca establecer una librería de software para resolver una clase grande de problemas de búsqueda en paralelo por el uso de numerosas computadoras alrededor del mundo con acceso a Internet. Se presentan resultados preliminares de las corridas de prueba en [MOE2003].

Varios problemas de optimización combinatoria como el problema de empacamiento binario, el problema del clique máximo y el problema del agente viajero pueden ser modelados como problemas de asignación cuadrática (si en el problema de asignación cuadrática se hace que todos los flujos sean iguales a uno, el problema queda reducido al problema del agente viajero).

Se encuentra disponible en Internet una página llamada QAPLIB, en donde se encuentran las disertaciones, artículos, resultados y problemas de prueba del QAP más recientes, la dirección es:

http://www.opt.math.tu-graz.ac.at/qaplib/

Emplear instancias clásicas disponibles en Internet permite la comparación del desempeño de las técnicas, aún cuando el óptimo sea desconocido. Para las instancias que proporciona la página del QAPLIB se tiene la mejor solución conocida o la solución óptima, ya sea porque se obtuvo ésta con algún algoritmo exacto o porque el problema proviene del generador de problemas con óptimo conocido diseñado por Li y Pardalos ([LI92]).

Recientemente se ha probado que tienen solución óptima algunas instancias del problema de asignación cuadrática: Bur26 (de la b a la h) (2004) y Tai25a (2003) por Hahn; Ste36a (2001) por Brixius y Anstreicher; Bur26a (2001) por Hahn; Kra30a (2000) por Hahn et al.; Kra30b, Kra32 y Tho30 (noviembre 2000) por Anstreicher, Brixius, Goux y Linderoth; Nug30 (2000) por Anstreicher, Brixius, Goux y Linderoth; Ste36b y Ste36c (1999) por Nyström.

En diciembre de 2000, Hahn encontró la solución óptima de la instancia Kra30b con el algoritmo descrito en [HAH2001], el tiempo de computadora empleado por Hahn et al. equivale a 182 días en una estación de trabajo con un cpu HP-3000, mientras que el tiempo que requirieron Anstreicher et al. en noviembre de 2000 para resolver la misma instancia equivale a 2.7 años en una estación de trabajo con un cpu HP-3000.

En 2004 Misevicius mejoró la mejor solución conocida de las instancias Tai50a, Tai80a y Tai100a usando una búsqueda tabú iterada.

Varios investigadores han buscado versiones particulares del problema de asignación cuadrática que se pueden resolver polinomialmente. Christofides y Gerrard ([CHR81]) estudiaron algunas instancias especiales del QAP; Sylla y Babu ([SYL87]) desarrollaron una metodología para un problema de asignación cuadrática ordenado; Chen ([CHE95]) al igual que Çela ([CEL98]) presentó otros casos del QAP que se pueden resolver polinomialmente; Herroelen y Van Gils ([HER85]), Cyganski et al. ([CYG94]), Mautor y Roucairol ([MAU94]) mostraron que las instancias del problema de asignación cuadrática proporcionadas por Palubetski son degeneradas; Angel y Zissimopoulos ([ANG98], [ANG2000], [ANG2001], [ANG2002]) discutieron la dificultad de otras instancias del QAP basados en la varianza de su flujo y distancia; Barvinok y Stephen ([BAR2003]) construyeron una distribución de los valores solución del QAP.

2.1. Cotas inferiores

Generalmente, los métodos exactos emplean enumeración implícita, en un intento por garantizar el óptimo y, al mismo tiempo, evitar la enumeración total de las soluciones factibles. El desempeño de estos métodos depende de la calidad computacional y de la eficiencia de las cotas inferiores.

Las cotas inferiores son herramientas fundamentales de las técnicas de ramificación y acotamiento y para la evaluación de la calidad de las soluciones

obtenidas de algunos algoritmos heurísticos. Se puede medir la calidad de una cota inferior por el trecho entre su valor y el de la solución óptima. Las cotas inferiores buenas deben estar cerca del óptimo. Las cotas inferiores son útiles para los métodos exactos, sólo cuando pueden ser calculadas rápidamente. Cuando se usan en métodos heurísticos, su calidad es más importante que su velocidad de cálculo.

La cota inferior para el QAP presentada por Gilmore ([GIL62]) y Lawler ([LAW63]) es una de las más conocidas. Su importancia se debe a su simplicidad y su bajo costo computacional. Sin embargo, presenta un inconveniente, su trecho crece muy rápido con el tamaño del problema, haciéndola una cota débil para instancias grandes. Las más recientes y prometedoras tendencias de investigación se basan en programación semidefinida, reformulación-linealización y técnicas de elevación-y-proyecto, aunque requieren un esfuerzo computacional extra. Anstreicher y Brixius ([ANS2001]) reportaron una nueva cota para el QAP usando programación semidefinida y programación cuadrática convexa con una buena relación entre costo y calidad.

La cota inferior de Gilmore y Lawler (GLB Gilmore Lawler Bound) está dada por la solución del siguiente problema de asignación lineal.

$$\begin{aligned}
&\text{Min} \sum_{i,j=1}^{n} (b_{ij} + l_{ij}) \cdot x_{ij} \\
&\text{s. a.} \quad \sum_{i=1}^{n} x_{ij} = 1 \quad 1 \leq j \leq n; \\
&\qquad \sum_{j=1}^{n} x_{ij} = 1 \quad 1 \leq i \leq n; \\
&\qquad x_{ij} \in \{0,1\} \quad 1 \leq i,\, j \leq n
\end{aligned}$$

Donde b_{ij} es el costo de asignar las instalaciones a las localizaciones. Para resolver el problema anterior es necesario calcular l_{ij}, $c_{ijkp} = f_{ij}d_{kp}$, f_{ij} es el flujo entre las facilidades i y j, y d_{kp} es la distancia entre las localizaciones k y p.

$$l_{ij} = \text{mín} \sum_{k,p=1}^{n} c_{ijkp} \cdot y_{ijkp} \qquad k \neq i,\, p \neq j$$

$$\text{s. a.} \quad \sum_{k=1}^{n} y_{ijkp} = 1 \quad 1 \leq i,\, j,\, p \leq n$$
$$\sum_{p=1}^{n} y_{ijkp} = 1 \quad 1 \leq i,\, j,\, k \leq n$$
$$y_{ijkp} \in \{0,1\} \quad 1 \leq i,\, j,\, k,\, p \leq n$$

Roucairol ([ROU79], [ROU87]), Frieze y Yadegar ([FRI83]), Burkard ([BUR91]), White ([WHI94]), Spiliopoulos y Sofianopoulou ([SPI98]) presentaron métodos mejorados para la GLB y sus aplicaciones a algoritmos usados para resolver el QAP.

Hay muchos métodos para calcular una cota inferior del problema de asignación cuadrática, sólo se hará referencia a los que se emplearon para determinar las cotas que aparecen en la figura 2.1 (la información que se presenta esta actualizada al 7 de marzo de 2010 y se extrajo de la página del QAPLIB).

Instancia	Óptimo	GL62	R-95	HG98	AB01	HH01	R-02	RS03	BV04	HZ07
Had16	3720	3358		3558	3595	3720*		3699	3672	3719**
Had18	5358	4776		5083	5143	5358*		5317	5299	5357**
Had20	6922	6166		6571	6677	6922*		6885	6811	6920
Kra30a	88900	68360	76003	75853	68572	86247		77647	86678	
Kra30b	91420	69065	76752	76562	69021	87107		81156	87699	
Nug12	578	493	523	523	498	578*	578*	557	568	577**
Nug15	1150	963	1041	1039	1001	1150*		1122	1141	1149**
Nug18	1930									1930*
Nug20	2570	2057	2182	2179	2290	2508		2451	2506	2568**
Nug22	3596					3511				3594**
Nug30	6124	4539	4805	4793	5365	5770		5803	5934	
Rou15	354210	298548	324869	323943	303777	354210*		333287	350207	354210*
Rou20	725520	559948	643346	642058	607822	699390		663833	695123	725314
Tai20a	703482	580674		617206	585139	675870		637300	671685	703482*
Tai25a	1167256	962417		1006749	983456	1091653		1041337	1112862	
Tai30a	1818146	1504688		1566309	1518059	1686290		1652186	1706875	
Tho30	149936	90578	100784	99995	124684	136708		136059	142814	

* Problema resuelto en forma exacta por el cálculo de una cota inferior

** Óptimo verificado por el cálculo de una cota

Figura 2.1: Comparación de cotas inferiores

En la figura 2.1, GL62 es la cota de Gilmore-Lawler ([GIL62]); R-95 es la cota de puntos interiores de Resende et al. ([RES95]); HG98 es la cota del dual ascendente de la técnica de reformulación-linealización de Hahn y Grant ([HAH98]); AB01 es la cota de programación cuadrática de Anstreicher y Brixius ([ANS2001]); HH01 es la cota Hahn-Hightower del dual ascendente de la técnica de reformulación-linealización nivel 2 de Adams et al. ([ADA2001]); R-02 es la cota de puntos interiores de la técnica de reformulación-linealización nivel 2 de Ramakrishnan et al. ([RAM2002]); RS03 es la cota de programación semidefinida de Rendl y Sotirov ([REN2003]); BV04 es la cota de la técnica de elevación-y-proyecto y de programación semidefinida de Burer y Vandenbussche ([BV2004]); HZ07 es la cota Hahn-Zhu de la técnica de reformulación-linealización nivel 3 de Zhu ([ZHU2007]). Las mejores cotas están en las celdas sombreadas.

El procedimiento dual ascendente de la columna HH1 estima por debajo la cota inferior de la técnica de reformulación-linealización nivel 2 descrita en [ADA2001] y en [RAM2002]. Asimismo, el procedimiento dual ascendente de la columna HG98 estima por debajo la cota de puntos interiores de [HAH98].

2.2. Métodos de solución

Los métodos usados en problemas de optimización combinatoria pueden ser exactos o heurísticos. En el primer caso, las estrategias que más se emplean son ramificación-y-acotamiento y métodos de programación dinámica.

2.2.1. Algoritmos exactos

Los diferentes métodos usados para alcanzar un óptimo global para el problema de asignación cuadrática incluyen ramificación-y-acotamiento, planos de corte o combinaciones de estos métodos, como ramificación-y-corte, y programación dinámica. Los algoritmos de ramificación-y-acotamiento son los más conocidos y utilizados. Hay varias referencias a algoritmos de ramificación-y-acotamiento para el problema de asignación cuadrática disponibles, como [NUR68], [GRA70], [BUR80], [PAR97], [BRI2001], [HAH2001], [ADA2001].

Procedimientos que combinan técnicas de ramificación-y-acotamiento con implementación en paralelo están siendo ampliamente usados. Debido a ello los mejores resultados del problema de asignación cuadrático están siendo

Tamaño	Cota	Año	Maquina	Velocidad del CPU	Segundos de CPU	Referencias
20	GL	1995	i860	40 MHz	811,440	Clausen y Perregaard ([CLA96], [CLA97])
20	RLT1	1995	SPARC10	75 MHz	159,900	Hahn et al. ([HAH98])
20	QP	1999	HP-C3000	300 MHz	8,748	Anstreicher y Brixius ([ANS2001])
20	RLT1	1999	UltraSPARC10	360 MHz	5,087	Hahn et al. ([HAH2001])
22	C-M	1995	i860	40 MHz	48,308,400	Clausen y Perregaard ([CLA96], [CLA97])
22	RLT1	1995	DEC Alpha 500	300 MHz	1,812,420	Hahn et al. ([HAH98])
22	QP	1999	HP-C3000	300 MHz	8,058	Anstreicher y Brixius ([ANS2001])
22	RLT1	1999	UltraSPARC10	360 MHz	46,917	Hahn et al. ([HAH2001])
24	GL	1997	32 Motorola 604		82,252,800	Brüngger et al. ([BRU98])
24	RLT1	1997	DEC Alpha 500	300 MHz	4,859,940	Hahn ([HAH2000])
24	QP	2000	HP-C3000	300 MHz	349,794	Anstreicher y Brixius ([ANS2001])
24	RLT2	2000	DEC Alpha 500	300 MHz	1,487,724	Hahn et al. ([HAH2001])
25	RLT1	1998	UltraSPARC10	360 MHz	5,698,818	Hahn ([HAH2000])
25	QP	2000	HP-C3000	300 MHz	715,020	Anstreicher et al. ([ANS2002])
25	RLT1	2000	HP-J5000	440 MHz	1,393,117	Hahn et al. ([HAH2001])
25	RLT2	2000	Dell 7150	733 MHz	254,179	Adams et al. ([ADA2001])
27	QP	2000	HP-C3000	300 MHz	5,676,480	Anstreicher et al. ([ANS2002])
27	RLT2	2001	IBM S80	450 MHz	1,579,956	Adams et al. ([ADA2001])
28	QP	2000	HP-C3000	300 MHz	27,751,680	Anstreicher et al. ([ANS2002])
28	RLT2	2001	IBM S80	450 MHz	8,682,044	Adams et al. ([ADA2001])
30	QP	2000	HP-C3000	300 MHz	218,859,840	Anstreicher et al. ([ANS2002])
30	RLT2	2004	Dell 7150	733 MHz	59,986,514	Adams et al. ([ADA2001])

GL - Gilmore-Lawler, RLT - Técnica de reformulación-linealización nivel 1, QP - Programación cuadrática,
C-M - Carraresi-Malucelli ([CAR92], [CAR93]), RLT2 - Técnica de reformulación-linealización nivel 2.

Figura 2.2: Progresos realizados al resolver en forma exacta instancias de Nugent

alcanzados. El éxito en las instancias de tamaños más grandes también está relacionado con las mejoras tecnológicas de hardware. En la figura 2.2 se presenta un resumen de lo que la metodología y las mejoras en la velocidad de las computadoras alcanzó entre 1995 y 2004, usando como ejemplo el progreso alcanzado al resolver en forma exacta instancias clásicas de Nugent ([NUR68]). Los datos provienen de Hahn ([HAH2000], [HAH2001]) y Brixius y Anstreicher ([BRI2001]). Todos los resultados que se presentan en la figura 2.2 se obtuvieron de variaciones del método de ramificación-y-acotamiento.

La programación dinámica es una técnica usada para casos del problema de asignación cuadrática en donde la matriz de flujo es la matriz de adyacencia de un árbol. Christofides y Benavent ([CHR89]) estudiaron este caso

usando una aproximación de programación lineal entera mixta al problema relajado. La cual resolvieron con un algoritmo de programación dinámica, tomando ventaja de la complejidad polinomial de estas instancias. Urban ([URB98]) también uso esta técnica.

El método de planos de corte introducido por Bazaraa y Sherali ([BAZ80]) no presentó resultados satisfactorios. Sin embargo, contribuyó en la formulación de algunas heurísticas que usan programación lineal entera mixta y descomposición de Benders. La lenta convergencia de este método lo hace propio sólo para instancias pequeñas ([KAU78], [BAZ80], [BAZ82]). Miranda et al. ([MIR2005]) usaron un algoritmo de descomposición de Benders en un problema de diseño de la placa base que incluyó costos lineales en la formulación.

Una variación a la técnica de ramificación-y-corte fue propuesta por Padberg y Rinaldi ([PAD91]) en esta estrategia alternativa de corte se explota el politopo (región finita de un espacio n-dimensional encerrado por un número finito de hiperplanos) definido por las soluciones factibles del problema. Su principal ventaja sobre los planos de corte es que los cortes están asociados con facetas del politopo. Los cortes asociados con facetas son más efectivos que los producidos por planos de corte, así la convergencia a una solución óptima se acelera. El escaso conocimiento sobre el politopo del problema de asignación cuadrática es la razón por la cual los planos de corte poliédricos no son ampliamente usados para este problema. Algunos investigadores han estado describiendo las propiedades básicas del politopo que pueden contribuir al futuro desarrollo de algoritmos: Padberg y Rijal ([PAD96]), Kaibel ([KAI98]), Jünger y Kaibel ([JUN2000], [JUN2001a], [JUN2001b]).

2.2.2. Algoritmos heurísticos

Los algoritmos heurísticos no dan garantía de optimalidad para la mejor solución obtenida. Los métodos aproximados se incluyen en esta categoría. Se llama metaheurísticas a las técnicas que pueden ser adaptadas a un amplio rango de problemas de optimización.

Los métodos constructivos fueron introducidos por Gilmore ([GIL62]). Hay varias referencias a algoritmos que emplean heurísticas de construcción, por ejemplo, Armour y Buffa ([ARM63]), Sarker et al. ([SAR95]), Gutin y Yeo ([GUT2002]). A finales de 1990, técnicas multi inicio se usaron para comenzar métodos heurísticos y metaheurísticos, en esta categoría se puede citar a los artículos de Misevicius ([MIS97]), Fleurent y Glover ([FLE99]) y

Misevicius y Riskus ([MIS99]).

Los métodos de mejoramiento corresponden a algoritmos de búsqueda local. La mayor parte de las heurísticas empleadas en el problema de asignación cuadrática están en esta categoría ([HEI73], [BRU84], [AND96], [DEI2000]). Las metaheurísticas frecuentemente usan métodos de mejoramiento.

2.2.3. Metaheurísticas

Antes del final de la década de 1980, la mayor parte de los métodos heurísticos propuestos para problemas de optimización combinatoria eran específicos y dedicados a un problema dado. Después de ese periodo, esto cambió. Aparecieron las metaheurísticas que son unas técnicas más generales. Se caracterizan por la definición de una estrategia a priori adaptada a la estructura del problema. Varias de estas técnicas están basadas en alguna forma de simulación de un proceso natural estudiado dentro de otro campo de conocimiento.

Las siguientes metaheurísticas se basan en metáforas de un proceso natural

Recocido simulado es un algoritmo de búsqueda local que explota la analogía entre los algoritmos de optimización combinatoria y la mecánica estadística (Kirkpatrick et al. [KIR83]). Esta analogía se hace mediante la asociación de las soluciones factibles del problema de optimización combinatoria con los estados de un sistema físico, teniendo costos asociados a estos estados de energía. Sean E_i y E_{i+1} dos estados de energía sucesivos, correspondientes a dos soluciones vecinas y sea $\Delta E = E_{i+1} - E_i$. Las siguientes situaciones pueden ocurrir: si $\Delta E < 0$, hay una reducción de energía y el proceso continua, es decir, hay una reducción en la función de costo del problema y la nueva asignación puede aceptarse; si $\Delta E = 0$, no hay cambio en el estado de energía, esto es, la función de costo del problema no cambio; si $\Delta E > 0$, hay un incremento en la energía y es útil para el proceso físico permitir un acomodo de partículas, en este caso, la función de costo del problema se incrementa. En lugar de eliminar esta asignación, ésta puede o no ser aceptada, de acuerdo al valor de una función de probabilidad, para evitar la convergencia a mínimos locales pobres. Burkard y Rendl ([BUR84]) propusieron una de las primeras aplicaciones de recocido simulado al problema de asignación cuadrática. Wilhelm y Ward ([WIL87]) presentaron nuevos

componentes de equilibrio para el recocido simulado. Connolly ([CON90]) presentó un concepto de "temperatura óptima" que dio resultados valiosos. Yip y Pao ([YIP94]), Mavridou y Pardalos ([MAV97]), Tian et al. ([TIA99]) y Misevicius ([MIS2003]) presentaron otros enfoques de recocido simulado aplicado al problema de asignación cuadrática.

Algoritmos genéticos son técnicas que simulan la selección natural y adaptación que se encuentran en la naturaleza. Estos algoritmos mantienen una población formada por un subconjunto de personas que corresponden, en el caso del problema de asignación cuadrática, a permutaciones factibles, con valores asociados al costo de la permutación. Por medio de *operadores genéticos*, y de un *criterio de selección*, el algoritmo reemplaza una población por otra con los mejores valores asociados disponibles. El esquema está basado en la idea de que los mejores individuos sobreviven y generan descendientes que transmiten sus características genéticas, del mismo modo en que las espacies biológicas se propagan en la naturaleza.

Los algoritmos genéticos generalmente comienzan con una población generada aleatoriamente. Sus costos se evalúan y un subconjunto con los menores costos es seleccionado. Los operadores genéticos son aplicados a este subconjunto, así se genera un nuevo subconjunto solución (o nueva población). El proceso continua hasta que algún criterio de terminación se cumple. Ver Davis ([DAV87]) y Goldberg ([GOL89]). Varios investigadores emplearon algoritmos genéticos para resolver el problema de asignación cuadrática, algunos de ellos son Tate y Smith ([TAT95]), Mavridou y Pardalos ([MAV97]), Kochhar et al. ([KOC98]), El-Baz ([ELB2004]) y Drezner ([DRE2005]).

El uso de estos algoritmos para resolver el problema de asignación cuadrática presenta algunas dificultades para obtener soluciones óptimas, aún en instancias pequeñas. Sin embargo, algunas ideas hibridas que emplean algoritmos genéticos han mostrado ser más eficientes.

Búsqueda dispersa es una técnica que presentó Glover ([GLO77]) en un estudio heurístico de problemas de programación lineal entera. Es un método evolutivo que toma combinaciones lineales de un subconjunto de la población para producir nuevas soluciones. Esta metaheurística está compuesta de una fase inicial y otra evolutiva. En la fase inicial se crea un conjunto referencia de buenas soluciones. En cada fase evolutiva posterior, nuevas soluciones son generadas usando combinaciones estratégicamente seleccionadas del sub-

conjunto referencia. Entonces un conjunto de las mejores soluciones recién generadas es trasladado al conjunto referencia. El proceso de la fase evolutiva se repite hasta que se satisface un criterio de terminación. En [CUN97] Cung et al. aplicaron búsqueda dispersa al problema de asignación cuadrática.

Optimización por colonia de hormigas se refiere a una clase de algoritmos distribuidos cuya característica más importante es la definición de las propiedades en la interacción de varios agentes simples (las hormigas).Se basa en el modo en que las hormigas están en posibilidad de encontrar un camino de la colonia a la comida. El conjunto de hormigas cooperando en una actividad ordinaria para resolver un problema, constituye el sistema de las hormigas. La principal característica de este método es el hecho de que la interacción de estos agentes genera un efecto sinérgico, porque la calidad de las soluciones obtenidas se incrementa cuando estos agentes trabajan juntos, interactuando entre ellos. Dorigo et al. ([DOR96]), Gambardella et al. ([GAM99]), y Maniezzo y Colorni ([MAN99]) presentaron resultados numéricos que muestran que la metaheurística de la colonia de hormigas es competitiva. Hay varias referencias más, las más sobresalientes son las siguientes: Stützle y Dorigo ([STU99]), Stützle y Hoos ([STU2000]), Talbi et al. ([TAL2001]) y Middendorf et al. ([MID2002]).

Redes neuronales y cadenas de Markov pese a que son diferentes de las metaheurísticas, son una metáfora de la naturaleza y han sido aplicadas al problema de asignación cuadrática. Sólo se mencionan los artículos más importantes en los que se emplearon estas. Utilizaron redes neuronales en los artículos de Liang ([LIA96]), Tsuchiya et al. ([TSU96]), Ishii y Sato([ISH98], [ISH2001]) y Hasegawa et al. ([HAS2002]). Aplicaron ambos conceptos en el artículo de Uwate et al. ([UWA2004]).

Las siguientes metaheurísticas se basan en consideraciones teóricas y experimentales

La búsqueda tabú es un algoritmo de búsqueda local que presentó Glover ([GLO89], [GLO90]) para encontrar soluciones de buena calidad a problemas de programación entera. Su principal característica es una lista actualizada de las mejores soluciones que se encontraron en el proceso de búsqueda. Cada solución recibe un valor de prioridad. Los elementos básicos de esta metaheurística son: una *lista tabú*, que se usa para guardar la historia de la

evolución del proceso de búsqueda; un mecanismo que permite la aceptación o rechazo de una nueva asignación en la vecindad, basado en la información de la lista tabú y en sus prioridades; y un mecanismo que permite la alternancia entre diversificación de vecinos y estrategias de intensificación. Algunos de los investigadores que emplearon búsqueda tabú para resolver el problema de asignación cuadrática son los siguientes: Skorin-Kapov ([SKO90], [SKO94]), Taillard ([TAI91], [TAI95]), Bland y Dawson ([BLA91]) y Chakrapani y Skorin-Kapov ([CHA93]).

El procedimiento de búsqueda voraz aleatorio adaptativo (GRASP, Greedy Randomized Adaptive Search Procedure) es una técnica iterativa que en cada paso obtiene una solución aproximada para el problema. En cada paso se construye una solución a través de un procedimiento voraz aleatorio adaptativo, las siguientes soluciones se obtienen aplicando a la solución anterior un algoritmo de búsqueda local que trata de generar una nueva solución mejor que la anterior. Esta técnica fue aplicada para resolver el problema de asignación cuadrática por varios investigadores, algunos de los cuales son los siguientes: Li et al. ([LI94]), Feo y Resende ([FEO95]), Fleurent y Glover ([FLE99]) y Ahuja et al. ([AHU2000]).

La búsqueda en vecindades variables (VNS, Variable Neighborhood Search) fue propuesta por Mladenovic y Hansen ([MLA97]). Está basada en movimientos sistemáticos dentro de un conjunto de vecinos, convenientemente definidos. Se pueden establecer varias reglas de cambio. Se efectúa un cambio cuando al explorar una vecindad no se produce una mejor solución. La búsqueda por vecindades variables se ha aplicado a instancias de problemas combinatorios grandes. En su artículo [TAI97], Taillard y Gambardella propusieron tres estrategias de búsqueda por vecindades variables para el problema de asignación cuadrática. Una de ellas es una búsqueda sobre una vecindad variable de acuerdo al modelo básico. Las otras dos son híbridos en combinación con alguno de los métodos antes descritos.

Hay muchos otros algoritmos híbridos para el problema de asignación cuadrática. Huntley y Brown ([HUN91]) y Bölte y Thonemann ([BOL96]) presentaron algoritmos que combinan recocido simulado y algoritmos genéticos. Battiti y Tecchiolli ([BAT94]), Chiang y Chiang ([CHI98]) y Talbi et al. ([TAL98]) emplearon búsqueda tabú y recocido simulado. Hasegawa et

al. ([HAS2002]) usaron búsqueda tabú con redes neuronales. Fleurent y Ferland ([FLE94]), y Drezner ([DRE2003]) combinaron algoritmos genéticos y búsqueda tabú. Mientras que Ahuja et al. ([AHU2000]) aplicaron un algoritmo genético que incorpora muchos principios de los algoritmos voraces.

Recientemente se han presentado algoritmos híbridos más complejos. Lim et al. ([LIM2000], [LIM2002]) propusieron algoritmos genéticos híbridos basados en búsqueda local por intercambio de k-genes. Balakrishnan et al. ([BAL2003]) y Misevicius ([MIS2004]) introdujeron algoritmos genéticos híbridos. Dunker et al. ([DUN2004]) combinaron programación dinámica y búsqueda genética.

Algunos algoritmos genéticos híbridos son conocidos como algoritmos meméticos o evolutivos, este es el caso de los algoritmos implementados por: Nissen ([NIS94], [NIS97]), Merz y Freisleben ([MER97], [MER99], [MER2000]) y Huntley y Brown ([HUN96]), entre otros.

Por último, se consideran de interés las investigaciones realizadas por Marinezzo y Colorni ([MAN95]) y Taillard et al. ([TAI2001]), ellos analizaron varias metaheurísticas las aplicaron al problema de asignación cuadrática y compararon sus resultados.

Capítulo 3

Algoritmo de solución

La Búsqueda Dispersa 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 ([GLO77]).

Una descripción general del método de Búsqueda Dispersa es la siguiente: se genera un conjunto de soluciones diversas D (alrededor de 100) del que se extrae un **conjunto referencia** que contiene b soluciones, con b entre 10 y 20. El conjunto referencia inicial contiene las $b/2$ mejores soluciones del conjunto D. Las $b/2$ restantes se extraen de D por el criterio de máxima distancia con las ya incluidas en el conjunto referencia, para ello se emplea una función de distancia, la cual mide la distancia entre las soluciones. Como la Búsqueda Dispersa se basa en combinar las soluciones del conjunto de referencia, se consideran subconjuntos de 2 o más soluciones del conjunto de referencia y se combinan mediante una rutina diseñada para tal efecto. La solución o soluciones que se obtienen de esta combinación pueden ser inmediatamente introducidas en el conjunto de referencia (actualización dinámica) o almacenadas temporalmente en una lista hasta terminar de realizar todas las combinaciones y después determinar qué soluciones entran en éste (actualización estática). Las soluciones combinadas pueden entrar en el conjunto referencia y reemplazar a algunas de las ya incluidas si las mejoran. Así, el conjunto referencia mantiene un tamaño b constante, pero va mejorando a lo largo de la búsqueda. Asimismo, el método de Búsqueda Dispersa contiene un **método de mejora**, esto es, un método de búsqueda local para mejorar las soluciones que genera, tanto las del conjunto D, como las soluciones combinadas. El proceso de combinación se repite hasta que se satisface un criterio de terminación.

```
Búsqueda Dispersa (Ver 3.1.4)
  Se genera la población inicial (Ver 3.1.3)
  k:=0
  Mientras no se cumpla la condición de término (ver 3.1.4) hacer
    Inicio
      - k:=k+1
      - si (k mod 9)=8 entonces
           Intensificación (Ver 3.1.6)
        en otro caso
           si (k mod 9)=0 entonces
              Diversificación (Ver 3.1.7)
           en otro caso
              Combinar soluciones (Ver 3.1.5)
    Fin
```

Figura 3.1: Algoritmo de Búsqueda Dispersa en pseudo-código

El algoritmo heurístico que se implementó y se describe a continuación está basado en el algoritmo de Búsqueda Dispersa que presentaron Cung et al. en 1997 (ver [CUN97]), con algunas diferencias que se irán señalando (por ejemplo: se agregó el procedimiento "mejora" al algoritmo presentado por Cung, lo que hace "mejora" es obtener soluciones de mayor calidad en el conjunto referencia). En la figura 3.1 se presenta el algoritmo de búsqueda dispersa en pseudo-código.

3.1. Descripción del algoritmo

En el capítulo anterior se mencionó que para resolver el Problema de Asignación Cuadrática se necesita encontrar una permutación π del conjunto $N = \{1, 2, ..., n\}$ que minimice el valor de la función objetivo:

$$f(\pi) = \sum_{i=1}^{n} \sum_{k=1}^{n} a_{ik} b_{\pi(i)\pi(k)}$$

Para realizar algunos cálculos se requiere de la matriz de permutación asociada a la permutación π, si $\pi = (\pi(1), \pi(2), ..., \pi(n))$, la matriz de permutación asociada a π es una matriz X de $n \times n$ tal que $x_{ik} = 1$ si $k = \pi(i)$ y $x_{ik} = 0$ en otro caso, $x_{ik} = 1$ significa que el edificio k se asigna

al sitio i. La matriz de permutación satisface las siguientes igualdades.

$$\sum_{i=1}^{n} x_{ik} = 1 \text{ para } k = 1, \ldots, n \quad \text{y} \quad \sum_{k=1}^{n} x_{ik} = 1 \text{ para } i = 1, \ldots n$$

Para generar la población inicial se emplea el criterio de máxima distancia entre matrices de permutación en el conjunto E. Se define la distancia euclidiana entre las matrices de permutación X y Y como:

$$\|X - Y\| = \sqrt{\sum_{i=1}^{n}\sum_{k=1}^{n}(x_{ik} - y_{ik})^2}$$

Comienzo definiendo un par de procedimientos que usaré después.

3.1.1. El operador

Dada una solución factible (permutación) U, el objetivo del operador es mejorar el costo de ésta, para ello se empleará un método de Búsqueda Tabú:

1. Un movimiento consiste en intercambiar los edificios localizados en los sitios i y j, al realizarlo se obtiene una nueva permutación, la cual se evalúa en la función objetivo, por lo que se dice que realizar un movimiento equivale a revisar un vecino.

2. La vecindad de un punto U se define como el conjunto de soluciones que pueden ser alcanzadas en un movimiento desde U.

3. Una vez que un movimiento es realizado, el movimiento opuesto se considera tabú por un número S de iteraciones.

Dada una permutación U el operador revisa: 80 vecinos distintos de U si $n \leq 20$; ó 200 si $n > 20$. Al final el operador devuelve al vecino V que tiene el menor valor de la función objetivo (puede ocurrir que el operador devuelva a U mismo). Para satisfacer la condición 3, al efectuar un movimiento en U, se eligen los sitios i y j de manera que $i < j$, se guarda en una lista los movimientos realizados y no se permite realizar dos veces el mismo movimiento.

3.1.2. El procedimiento "mejora"

En el algoritmo que se implementó E es el **conjunto referencia,** se estableció que el número de elementos de E sería n si $n \leq 20$, ó 20 si $n > 20$, en el artículo de Cung ([CUN97]) la cardinalidad de E es n.

Dependiendo del valor de n, este procedimiento revisa ve vecinos de cada permutación en E y deja en su lugar la permutación con menor valor de la función objetivo.

Si $n \in \{19, 20\}$, $ve = 80$.

Si $n \in \{21, ..., 90\}$, $ve = 200$.

Si $n \in \{91, ..., 150\}$, $ve = 250$.

Este procedimiento recibe un parámetro rep (repeticiones), si $rep = 0$ se repite el proceso anterior m veces, dependiendo del valor de n.

Si $n = 19$, $m = 14$.

Si $n \in \{20, ..., 26\}$, $m = 16$.

Si $n \in \{27, ..., 50\}$, $m = 30$.

Si $n \in \{51, ..., 89\}$, $m = 50$.

Si $n \in \{90, ..., 100\}$, $m = 250$.

Si $n \in \{101, ..., 150\}$, $m = 400$.

Pero si $rep > 0$, entonces $m = rep$, es decir, se repite el proceso anterior $m = rep$ veces. Se deja a los elementos de E ordenados de mejor a peor.

3.1.3. Población inicial

Se emplearán permutaciones circulares, así que se las va a definir. Se llama permutaciones circulares al número de formas en que se pueden ordenar n objetos distintos alrededor de un círculo, hay $(n-1)!$ permutaciones circulares (ver [JOH97, p. 212]). Para $n = 3$ el número de permutaciones circulares distintas es $2! = 2$, por ejemplo, las que se muestran en la figura 3.2 ((1, 2, 3), (1. 3, 2)), son permutaciones circulares distintas.

Dos permutaciones circulares son equivalentes si una de ellas se obtiene a partir de la otra mediante un giro. Así las dos primeras permutaciones que se presentan en la figura 3.3 ((2, 3, 1), (3, 1, 2)) son equivalentes a la primera

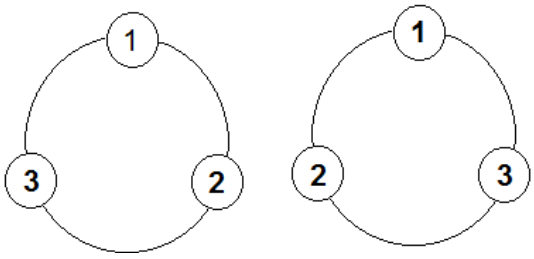


Figura 3.2: Permutaciones circulares distintas

permutación de la figura 3.2 ((1, 2, 3)) y las dos últimas permutaciones de la figura 3.3 ((3, 2, 1), (2, 1, 3)) son equivalentes a la segunda permutación de la figura 3.2 ((1, 3, 2)).

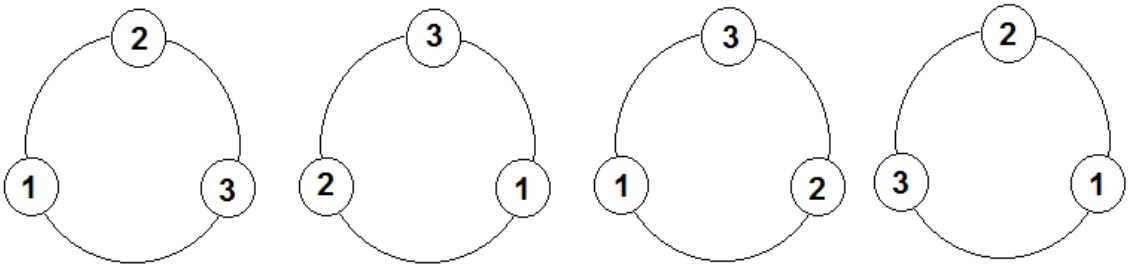


Figura 3.3: Permutaciones

Dada una permutación circular de tamaño n, hay $n-1$ permutaciones equivalentes a ella, es decir, cada permutación circular de n objetos equivale a n permutaciones distintas.

Ahora se puede empezar. La población inicial debe cubrir tanto como sea posible el espacio de soluciones, de manera que sea posible alcanzar el óptimo donde sea que éste se encuentre. Así que la población inicial debe ser tan dispersa como sea posible. Esto puede alcanzarse generando una población que maximice la distancia euclidiana entre cualquier par de puntos. Por tanto se debe tratar de generar puntos tales que la distancia entre cualesquiera dos puntos sea $\sqrt{2n}$ (es decir, la distancia máxima entre dos puntos factibles).Para ello se genera una permutación aleatoria P, a partir de ella se generan las $n-1$ permutaciones circulares equivalentes a P (la

distancia entre cualesquiera dos de las n permutaciones anteriores es $\sqrt{2n}$). Ejemplo, suponga $n = 4$ y que la permutación aleatoria que se generó es $P = (3, 1, 2, 4)$, las $n - 1 = 3$ permutaciones circulares equivalentes a P son:

$$(1, 2, 4, 3)$$
$$(2, 4, 3, 1)$$
$$(4, 3, 1, 2)$$

la distancia entre las matrices de permutación asociadas a dos cualesquiera de estas cuatro permutaciones es $\sqrt{8}$.

Después se genera otra permutación aleatoria Q, tal que la distancia entre P y Q sea mayor a cero (para asegurar que $Q \neq P$) y menor a $\sqrt{2n}$ (pues si $\|P - Q\| = \sqrt{2n}$ puede ocurrir que Q sea una permutación circular equivalente a P) y a partir de Q se generan las $n - 1$ permutaciones circulares equivalentes a Q. Luego se aplica a estas $2n$ permutaciones el operador y se dejan en E a las de menor valor, ordenadas de menor a mayor respecto al valor en la función objetivo. Finalmente se aplica el procedimiento "mejora" a la población inicial E con $rep = 0$ (así "mejora" realizará el número de rep que tiene establecido, de acuerdo al tamaño de n).

3.1.4. Búsqueda dispersa

Se agregó un criterio de terminación al algoritmo propuesto por Cung. Si se realizan $iteSinM = 23$ iteraciones sin que el conjunto E sea modificado o se llevan a cabo 500 iteraciones el algoritmo termina.

El valor de $iteSinM = 23$ se determinó de la siguiente manera. Se eligió una instancia de cada uno de los tamaños que había entre 26 y 49. Si había varias instancias del mismo tamaño se elegía la instancia en la que se obtuvo el mejor valor de la función objetivo. Se llevaron a cabo diez corridas de 500 iteraciones de estas instancias y se calculó el promedio de iteraciones sin modificación ($promISM$) de cada instancia. Primero se le dio a $iteSinM$ el valor promedio de $promISM$ de las instancias mencionadas, con ese valor el tiempo de corrida del algoritmo era grande, por lo que se decidió darle como valor a $iteSinM$ el mínimo del $promISM$ de las instancias en cuestión, así se obtuvo $iteSinM = 23$.

El algoritmo lleva a cabo 7 iteraciones ordinarias; una iteración de intensificación y una iteración de diversificación, este ciclo se repite hasta alcanzar la condición de término del algoritmo.

Adicionalmente en las iteraciones ordinarias el procedimiento de Búsqueda Dispersa llama al procedimiento "mejora":

- Si $n = 19$ se llama al procedimiento "mejora" con $rep = 10$ en las iteraciones 1, 19, 37,... $(1 + 18b$, con $b \in \{1, 2, 3, ...\})$.
- Si $n \in \{20, ..., 89\}$ se llama a "mejora" con $rep = 0$ en las iteraciones 1, 19, 37,... $(1+18b$, con $b \in \{1, 2, 3, ...\}$, cuando $rep = 0$, el procedimiento "mejora" realiza las iteraciones que de antemano tiene establecido que corresponde efectuar de acuerdo al tamaño de $n)$.
- Si $n = 90$ se llama a "mejora" con $rep = 45$ en las iteraciones 1, 28, 55,... $(1 + 27b$, con $b \in \{1, 2, 3, ...\})$.
- Si $n \in \{91, ..., 100\}$ se llama a "mejora" con $rep = 43$ en las iteraciones 1, 37, 73,... $(1 + 36b$, con $b \in \{1, 2, 3, ...\})$.
- Si $n \in \{101, ..., 150\}$ se llama a "mejora" con $rep = 0$ en la iteración uno (cuando $rep = 0$, el procedimiento "mejora" realiza las iteraciones que de antemano tiene establecido que corresponde efectuar de acuerdo al tamaño de $n)$.

3.1.5. Combinando soluciones

Una iteración ordinaria equivale a un llamado a este procedimiento, el cual combina soluciones.

Se elige aleatoriamente un número $j \in \{2, 3, 4, 5\}$ de acuerdo a una distribución uniforme. Nuevamente siguiendo una distribución uniforme se eligen j matrices de permutación en E, se suman y se guarda la suma en T. Cada matriz de permutación que se elige en E se suma a la matriz de frecuencias F, para que cada entrada de la matriz F reporte el número de veces que un edificio es asignado a un sitio, la matriz F se empleará en el procedimiento de diversificación. Sean $sitios = \{1, ..., n\}$ y $edificios = \{1, .., n\}$, aleatoriamente, siguiendo una distribución uniforme, se elige un sitio i en $sitios$, $sitios = sitios - \{i\}$. Elegir un sitio equivale a elegir un renglón de la matriz T. De entre las entradas con mayor valor en el renglón elegido de T, se elige aleatoriamente, siguiendo una distribución uniforme, un edificio j, $edificios = edificios - \{j\}$. Ahora $x_{ij} = 1$, de esta manera se obtiene una solución factible X (matriz de permutación) a partir de T. Se determina la

permutación correspondiente a X, se aplica el operador a la permutación y se obtiene como resultado una permutación V, se modifica E si el valor de la función objetivo en la permutación V es mejor que el peor elemento en E.

3.1.6. Intensificación

De acuerdo a lo establecido en la sección IV del artículo de Cung, el método de Búsqueda Tabú corre 1200 iteraciones en una iteración de intensificación del procedimiento de Búsqueda Dispersa. Sin embargo, el número de vecinos de un punto son las combinaciones de 2 en n, y son menos de 1200 para $n < 50$ (ver sección 3.1.1), por lo que no es posible correr 1200 iteraciones del método de Búsqueda Tabú para $n < 50$. Por consiguiente, se realizan procesos de intensificación distintos para $n < 50$ y $n \geq 50$.

- Si $n < 50$ y $n \in \{19, ..., 26\}$, se llama al procedimiento "Combinando soluciones", que produce la solución V con la cual se iniciara el procedimiento de intensificación, mientras que si $n \in \{27, ..., 49\}$ se iniciara la intensificación partiendo de la mejor solución del conjunto E. Partiendo del punto que corresponda se efectúa un movimiento y se obtiene un nuevo punto W, me muevo a W y efectúo un movimiento, de esta manera me voy moviendo de punto en punto y así llevo a cabo 1200 iteraciones. Se verifica en cada movimiento que los últimos 8 movimientos sean distintos, para cumplir la condición 3 del operador (ver sección 3.1.1) con $S = 8$. En cada movimiento se compara la solución obtenida con el peor elemento de la población. La nueva solución es incluida en la población si es mejor que el peor elemento. En el Cung no indican como realizan intensificación en un punto para el que $n < 50$.

- Si $n \geq 50$, se revisan 1200 vecinos de la mejor solución en E (en el artículo de Cung producen una solución combinada y revisan 1200 vecinos de esta, se parte de la mejor solución en E, porque se observó que de esta manera se obtenían mejores resultados) y se guarda en *menor* a la permutación con menor valor de la función objetivo. Para cumplir la condición 3 del operador con $S = 200$, se verifica en cada movimiento que los últimos 200 movimientos sean distintos (en el artículo de Cung toman $S = 200$ para $n > 90$). Aún con $S = 200$ se vio al revisar los 1200 vecinos generados que hay vecinos que se repiten. Después de revisar a todos los vecinos, se compara el valor de *menor* con el peor

elemento en la población, si el valor de *menor* es mejor que el del peor elemento, *menor* se incluye en la población.

3.1.7. Diversificación

Para tomar en cuenta la diversidad de la población algunos edificios (5 % de n) se asignan a sitios que se han usado poco, para ello se emplea la matriz de frecuencias F que se mencionó en la sección 3.1.5, la cual reporta en cada entrada el número de veces que un edificio es asignado a un sitio. Las asignaciones restantes se determinan con el procedimiento de la sección 3.1.5. El procedimiento que se sigue para asignar las 5 % de n entradas de la solución X es el siguiente:

Sean $sitios = \{1, ..., n\}$ y $edificios = \{1, .., n\}$, aleatoriamente, siguiendo una distribución uniforme, se elige un sitio i en $sitios$, $sitios = sitios - \{i\}$. Elegir un sitio equivale a elegir un renglón de la matriz F. De entre las entradas con menor valor en el renglón elegido de F, se elige aleatoriamente, siguiendo una distribución uniforme, un edificio j, $edificios = edificios - \{j\}$, se llega así a que $x_{ij} = 1$, procediendo de esta manera se obtiene una parte de la solución factible X el resto de X se determina con el procedimiento de la sección 3.1.5.

La solución de diversificación generada se combina con cada elemento en el conjunto E, se modifica E en caso de ser necesario. Lo anterior se agregó al procedimiento de diversificación establecido en el artículo de Cung. Con esta modificación se observa en los resultados que la diversificación funciona.

3.2. Experiencia computacional

El algoritmo de Búsqueda Dispersa fue implementado en Delphi 5, en una máquina con procesador Core 2 Quad a una velocidad de 2.40 GHz.

La prueba del algoritmo se llevo a cabo revisando todas las instancias de tamaño entre 26 y 49 que estaban en abril de 2010 en la sección de "Instancias y Soluciones"de la página del QAPLIB:

http://www.opt.math.tu-graz.ac.at/qaplib/

a esa fecha la página del QAPLIB estaba actualizada al 16 de febrero de 2009. Se eligió ese tamaño porque el caso real que se va a resolver es aproximadamente de ese tamaño. En 29 de las 38 instancias que se revisaron, esto

es, en el 76.31 % de los casos se llegó al óptimo o mejor solución conocida. Los resultados de estas corridas se muestran a continuación.

Problema	Mejor Solución Conocida		Mejor Solución Encontrada	Tiempo (s)
Bur26a	5,426,670	(óptima)	5,426,670	22.10
Bur26b	3,817,852	(óptima)	3,817,852	24.48
Bur26c	5,426,795	(óptima)	5,426,795	22.80
Bur26d	3,821,225	(óptima)	3,821,225	26.74
Bur26e	5,386,879	(óptima)	5,386,879	25.94
Bur26f	3,782,044	(óptima)	3,782,044	24.40
Bur26g	10,117,172	(óptima)	10,117,172	23.00
Bur26h	7,098,658	(óptima)	7,098,658	23.00
Nug27	5,234	(óptima)	5,234	39.80
Nug28	5,166	(óptima)	5,166	33.55
Kra30a	88,900	(óptima)	88,900	37.80
Kra30b	91,420	(óptima)	91,490	31.75
Lipa30a	13,178	(óptima)	13,178	38.06
Lipa30b	151,426	(óptima)	151,426	31.10
Nug30	6,124	(óptima)	6,124	38.99
Tai30a	1,818,146		1,845,252	39.12
Tai30b	637,117,113		637,117,113	43.77
Tho30	149,936		149,936	36.40
Esc32a	130		134	31.21
Esc32b	168		168	35.90
Esc32c	642		642	23.80
Esc32d	200		200	31.80
Esc32e	2	(óptima)	2	18.10
Esc32g	6	(óptima)	6	18.90
Esc32h	438		438	35.50
Kra32	88,700	(óptima)	88,700	33.47
Tai35a	2,422,002		2,462,488	40.64
Tai35b	283,315,445		283,315,445	46.44
Ste36a	9,526	(óptima)	9,612	39.69
Ste36b	15,852	(óptima)	15,852	37.76
Ste36c	8,239,110	(óptima)	8,239,110	42.94

Problema	Mejor Solución Conocida	Mejor Solución Encontrada	Tiempo (s)
Lipa40a	31,538 (óptima)	31,538	44.95
Lipa40b	476,581 (óptima)	476,581	44.60
Tai40a	3,139,370	3,189,222	44.47
Tai40b	637,250,948	637,250,948	49.04
Tho40	240,516	241,334	41.45
Sko42	15,812	15,830	46.26
Sko49	23,386	23,418	61.00

3.3. Descripción del algoritmo con restricciones

El Problema de Asignación Cuadrática considera que cualquier espacio puede ser asignado a cualquier sitio, sin embargo, en el caso real que se va a resolver esto no es posible porque los diferentes laboratorios y aulas (espacios) son de distintos tamaños. El algoritmo que se aplicará es el mismo, no obstante se sustituirán los procedimientos del algoritmo de Búsqueda Dispersa por los seis procedimientos con restricciones que se presentan más adelante.

Se tomará en cuenta que en lugar de que cualquier espacio pueda ser asignado a cualquier sitio, sólo se pueden intercambiar espacios del mismo tamaño, los n espacios se separan de acuerdo a su tamaño:

$$n = m_1 + m_2 + \ldots + m_k$$

donde m_1 son los espacios de tamaño T_1, m_2 son los espacios de tamaño T_2, y así sucesivamente. De esta manera se pasa de tener $n!$ soluciones factibles a tener $m_1! \cdot m_2! \cdot \ldots \cdot m_k!$. El problema sin restricciones tenía $40!=815.9\times10^{45}$ soluciones factibles, mientras que el caso con restricciones tiene $14!\times4!\times5!\times2!^8=64.27\times10^{15}$ soluciones factibles. En la primera tabla que se presenta en el siguiente capítulo se proporciona el área de los diferentes espacios, allí se observa que los 14 primeros espacios pueden intercambiar su lugar, que los siguientes 4 también pueden intercambiar sus lugares, los siguientes 5 también se pueden intercambiar y que los siguientes 14 sólo pueden intercambiar sus lugares de dos en dos, el laboratorio de Análisis Experimental es tan pequeño que no puede intercambiar su lugar con nadie, las entradas se consideraron como espacios ficticios (para que el modelo reflejara

```
Búsqueda Dispersa (Ver 3.3 y 3.1.4)
  Se genera la población inicial con restricciones (Ver 3.3.1)
  k:=0
  Mientras no se cumpla la condición de término (ver 3.1.4) hacer
    Inicio
      - k:=k+1
      - si (k mod 9)=8 entonces
           Intensificación con restricciones (Ver 3.3.5)
        en otro caso
           si (k mod 9)=0 entonces
              Diversificación con restricciones (Ver 3.3.6)
           en otro caso
              Combinar soluciones con restricciones (Ver 3.3.4)
    Fin
```

Figura 3.4: Algoritmo de Búsqueda Dispersa con restricciones en pseudo-código

el gran número de personas que entran al edificio toman clase y se van) y solamente pueden intercambiarse entre ellas.

Para ser consistente con lo que se escribió en la sección 3.1 seguiré llamando edificios a lo que al inicio de esta sección llame espacios.

En la figura 3.4 se describe el algoritmo de búsqueda dispersa con restricciones en pseudo-código. A continuación se presentan los procedimientos que se emplearán para resolver el problema con restricciones.

3.3.1. Población inicial con restricciones

Se quiere generar al menos $2n$ soluciones diversas y formar con las 20 mejores la población inicial o conjunto referencia, E.

Se hace $m_1 \geq m_i$, para toda $i = 1, \ldots k$. Se supondrá que $m_1! > 2n$ (si no fuera el caso lo que sí debe ocurrir es que el número de soluciones factibles sea mayor a $2n$), en el caso real se cumple este supuesto $14! > 2 \cdot 40$. Si $\frac{2n}{m_1}$ es entero se hace $j_1 = \frac{2n}{m_1}$, si no, se hace j_1 igual a la parte entera de $\frac{2n}{m_1} + 1$. Se genera una permutación aleatoria P_1 de tamaño m_1 y se llama al procedimiento "Una parte de la población inicial" (a partir de P_1, este procedimiento genera m_1 permutaciones de tamaño n), luego se repite lo que sigue desde $s = 2$ hasta $s = j_1$: se genera una permutación aleatoria

P_s tal que la distancia euclidiana entre la matriz que le corresponde y la matriz de permutación correspondiente a todas las P_t antes generadas (P_1, $P_2, \ldots, P_{s-1}$) sea mayor a cero (para que $P_s \neq P_t$) y menor a $\sqrt{2m_1}$ (para que P_s no sea una de las permutaciones circulares de P_t), y después se llama al procedimiento "Una parte de la población inicial".

Se aplica a estas $m_1 j_1$ permutaciones generadas el operador con restricciones, y se deja en el conjunto referencia E a las de menor valor, ordenadas de menor a mayor respecto al valor en la función objetivo.

Una parte de la población inicial

Este procedimiento recibe como entrada una permutación aleatoria P_1 de tamaño m_1 (es decir, las primeras m_1 entradas de la permutación 1, o de la que corresponda) y a partir de ella se generan las $m_1 - 1$ permutaciones circulares equivalentes a P_1 (cada una de las cuales nos da las primeras m_1 entradas de las permutaciones 2 a m_1, o de las permutaciones que correspondan).

Para $T = 2, 3$ llevar a cabo los puntos 1 y 2.

1. Se genera una permutación aleatoria Q_1 de tamaño m_T (esto es, las entradas $\sum_{p=1}^{T-1} m_p + 1$ a $\sum_{p=1}^{T} m_p$ de la permutación 1, o de la que corresponda). Si $\frac{m_1}{m_T}$ es entero $j_T = \frac{m_1}{m_T}$, si no, j_T es igual a la parte entera de $\frac{m_1}{m_T}$. Se repite el inciso (a) desde $i_T = 1, \ldots, j_T$.

(a) A partir de Q_{i_T} se generan $m_T - 1$ permutaciones circulares equivalentes a Q_{i_T}, cada una de las cuales da las entradas $\sum_{p=1}^{T-1} m_p + 1$ a $\sum_{p=1}^{T} m_p$ de las permutaciones $(i_T - 1)m_T + 2$ a $i_T m_T$ (o de las que corresponda). Se genera una permutación aleatoria Q_{i_T+1} cuya distancia euclidiana entre la matriz que le corresponde y la matriz de permutación correspondiente a todas las Q_h antes generadas ($Q_1, Q_2, \ldots, Q_{i_T}$) sea mayor a cero (para que $Q_{i_T+1} \neq Q_h$) y menor a $\sqrt{2m_T}$ (para que Q_{i_T+1} no sea una de las permutaciones circulares de Q_h).

2. Se generan las $m_1 - j_T m_T - 1$ permutaciones circulares equivalentes a Q_{j_T+1}, las cuales dan las entradas $\sum_{p=1}^{T-1} m_p + 1$ a $\sum_{p=1}^{T} m_p$ de las permutaciones $j_T m_T + 2$ a m_1 (o de las que corresponda). La permutación Q_{j_T+1}

da dichas entradas de la permutación $j_T m_T + 1$ (o de la permutación que corresponda).

Los siguientes 14 espacios sólo se pueden intercambiar de dos en dos, es decir, $m_i = 2$ para $i = 4, \ldots, 10$. Asimismo, las entradas (que son espacios ficticios) solamente se pueden intercambiar entre sí, $m_{12} = 2$. Para las permutaciones 1 a 14 (o las que corresponda)

- Siguiendo una distribución uniforme se elige un edificio de entre los dos que se pueden elegir para el sitio $2j$ y automáticamente el otro edificio elegible se asigna al sitio $2j + 1$, para $j = 12, \ldots, 18$.
- El edificio 38 es asignado al sitio 38, pues para este edificio esa es la única asignación posible.
- Siguiendo una distribución uniforme se elige una entrada de entre las dos que se pueden elegir para el sitio 39 y entonces la otra entrada es asignada al sitio 40.

3.3.2. El operador con restricciones

Dada una solución factible (permutación) U, el objetivo del operador es mejorar el costo de ésta, para ello se empleara este método:

1. Un movimiento consiste en intercambiar los edificios localizados en los sitios i y j, al realizarlo se obtiene una nueva permutación, la cual se evalúa en la función objetivo. Ahora sólo se pueden intercambiar edificios del mismo tamaño, así que si el tamaño del edificio i es T_x, j debe elegirse dentro del conjunto de edificios de tamaño T_x.
2. La vecindad de un punto U se define como el conjunto de soluciones que pueden ser alcanzadas en un movimiento desde U.
3. Una vez que un movimiento es realizado, el movimiento opuesto se considera tabú por un número S de iteraciones.

Si sólo hay un edificio de un cierto tamaño, ese edificio es fijo. Se separó a los n edificios de acuerdo a su tamaño en $m_1, \ldots, m_k$, $n = m_1 + \ldots + m_k$.

El número de vecinos de un punto U es igual a:

$$veci = \sum_{\substack{j=1 \\ m_j \neq 1}}^{k} \binom{m_j}{2}$$

$\binom{m_j}{2}$ son las combinaciones de m_j elementos tomados de dos en dos. Dada una permutación U el operador revisa máx$\{80, veci\}$ vecinos distintos de U si $n \leq 20$ ó máx$\{200, veci\}$ si $n > 20$. En el caso real máx$\{200, veci\} = veci = 115$. Al terminar el operador devuelve al vecino W con el menor valor de la función objetivo, a veces el operador devuelve a U mismo.

3.3.3. El procedimiento mejora con restricciones

Como se dijo en la subsección anterior se separo a los n edificios por su tamaño en $m_1, \ldots, m_k$, $n = m_1 + \ldots + m_k$. El número de vecinos de un punto U es igual a:

$$veci = \sum_{\substack{j=1 \\ m_j \neq 1}}^{k} \binom{m_j}{2}$$

Dependiendo del valor de n este procedimiento revisa ve vecinos de cada permutación en el conjunto referencia y deja en su lugar a la permutación con menor valor de la función objetivo.

Si $n = 20$, $ve =$ máx$\{80, veci\}$

Si $n \in \{21, \ldots, 90\}$, $ve =$ máx$\{200, veci\}$

Si $n \in \{91, \ldots, 150\}$, $ve =$ máx$\{250, veci\}$

Si al revisar los vecinos de una permutación del conjunto referencia el edificio elegido i es de tamaño T_z,el edificio j con el que el edificio i puede intercambiarse deberá ser elegido de entre los edificios de tamaño T_z.

Este procedimiento recibe un parámetro rep (repeticiones).

- Si $rep = 0$

  Dependiendo del valor de n, se repite el proceso anterior m veces.

  Si $n \in \{20, ..., 26\}$, $m = 16$.

Si $n \in \{27, ..., 50\}$, $m = 30$.

Si $n \in \{51, ..., 89\}$, $m = 50$.

Si $n \in \{90, ..., 100\}$, $m = 250$.

Si $n \in \{101, ..., 150\}$, $m = 400$.

- Si $rep > 0$

  $m = rep$ es el número de veces que se repetirá el proceso anterior.

Se deja a los elementos del conjunto referencia ordenados de mejor a peor.

3.3.4. Combinando soluciones con restricciones

Se elige aleatoriamente un número $j \in \{2, 3, 4, 5\}$ de acuerdo a una distribución uniforme. Nuevamente siguiendo una distribución uniforme se eligen j matrices de permutación en E, se suman y se guarda la suma en T. Cada matriz de permutación que se elige en E se suma a la matriz de frecuencias F, la matriz F se empleará en el procedimiento de diversificación. Sean $sitios = \{1, ..., n\}$, se repite el procedimiento del párrafo que sigue hasta que $sitios = \varnothing$.

Aleatoriamente, siguiendo una distribución uniforme, se elige un sitio i en $sitios$, si el tamaño del sitio es T_j,se determinan $m_j \times m_j$ entradas de la matriz de permutación X, que corresponden a m_j entradas de la solución combinada (permutación), es decir, se genera la parte de la solución combinada de los sitios que se pueden intercambiar, por ser todos de tamaño T_j, $sitios = sitios - sitios\_de\_tamaño\_T_j$.

Se determina la permutación correspondiente a la matriz de permutación X, se aplica el operador a la permutación y se obtiene como resultado una permutación V, se modifica E si el valor de la función objetivo en la permutación V es mejor que el peor elemento en E.

3.3.5. Intensificación con restricciones

Como el caso real a resolver es de tamaño 40, sólo se modificó la intensificación para el caso $n < 50$. Si $n \in \{19, \ldots, 26\}$ se llama al procedimiento "Combinando soluciones con restricciones" y se inicia la intensificación partiendo de la solución V que genera dicho procedimiento, pero si

$n \in \{27, \ldots, 49\}$ se inicia la intensificación partiendo de la mejor solución del conjunto referencia. Partiendo de la solución que corresponda se efectúa un movimiento, si el tamaño del edificio i que se elige siguiendo una distribución uniforme es T_x, j debe elegirse, siguiendo también una distribución uniforme, dentro del conjunto de edificios de tamaño T_x, se obtiene una nueva solución W, me muevo a W y efectúo un movimiento, de esta manera se lleva a cabo 1200 iteraciones. Se verifica en cada movimiento que los últimos 8 movimientos sean distintos. En cada movimiento se compara la solución obtenida con el peor elemento de la población, si es mejor la nueva solución sustituye al peor elemento.

3.3.6. Diversificación con restricciones

Para tomar en cuenta la diversidad de la población algunos edificios (5 % de n) se asignan a sitios que se han usado poco, para lo cual se emplea la matriz de frecuencias F que se mencionó en la sección 3.3.4, la cual reporta en cada entrada el número de veces que un edificio es asignado a un sitio. El resto de las entradas se determinan con el procedimiento de la sección 3.3.4.

La parte del espacio de soluciones en donde tiene sentido buscar que sitios se han usado poco es en los primeros 14 espacios, así que es allí donde se busca. Sean $sitios2 = \{1, \ldots, 14\}$ y $edificios2 = \{1, \ldots, 14\}$. Aleatoriamente, siguiendo una distribución uniforme, se elige un sitio i en $sitios2$, $sitios2 = sitios2 - \{i\}$. La matriz de frecuencias F es una matriz cuadrada de tamaño n que en la diagonal tiene matrices cuadradas de tamaños $m_1, m_2, \ldots, m_k$, así en la esquina superior derecha de F está una matriz cuadrada de tamaño m_1, mientras que en la esquina inferior izquierda de F está una matriz cuadrada de tamaño m_k. Elegir un sitio equivale a elegir un renglón de la matriz F, como se quiere que el sitio i sea asignado a un sitio del mismo tamaño que i, se toman del renglón de la matriz F las entradas que están entre 1 y 14, y se ordenan de menor a mayor, de entre las de menor valor se elige aleatoriamente, siguiendo una distribución uniforme, un edificio j, $edificios2 = edificios2 - \{j\}$, así se determina la entrada $x_{ij} = 1$ de la matriz de permutación. Procediendo de esta manera se determina el 5 % de los edificios, lo cual corresponde a determinar una parte de la matriz de permutación X, el resto de X se determina con el procedimiento de la sección 3.3.4

La solución de diversificación generada se combina con cada elemento del conjunto referencia, se modifica a este último en caso de ser necesario.

Capítulo 4

Aplicación a un problema real

Uno de los edificios de la Facultad de Ingeniería de la UNAM es el edificio "Alberto Camacho", se tiene el proyecto de construir un nuevo edificio para éste que se llamará Centro de Ingeniería Avanzada "Alberto Camacho" (CIA). El problema a resolver es encontrar la localización de los espacios dentro del edificio del CIA que minimice las distancias que se recorren dentro del mismo.

Las personas involucradas en el proyecto de construcción del CIA me proporcionaron: un archivo en Excel con los "requerimientos de espacios arquitectónicos", 24 cuestionarios (los cuales fueron llenados por los encargados de los diferentes laboratorios) y los planos en AutoCAD de un edificio para el CIA (ver figuras 4.1, 4.2 y 4.3). Se compararon los cuestionarios con el archivo de requerimientos de espacios, no se encontraron diferencias que valga la pena mencionar. El archivo de requerimientos de espacio no coincide con los planos que me proporcionaron, se comentó esto con las personas involucradas en el proyecto, ellos dijeron que los planos son de una propuesta anterior a los requerimientos de espacios proporcionados, lo que ocurre es que han seguido avanzando en la determinación de sus requerimientos reales de espacio.

Se decidió, junto con los responsables del proyecto, emplear los planos del edificio y proporcionar la metodología, para que ellos la puedan aplicar al edificio que finalmente se vaya a construir. La persona que designaron para atenderme me ayudó a localizar en los planos la mayor parte de los espacios que aparecen en el archivo de "requerimientos de espacios arquitectónicos".

En el archivo de "requerimientos de espacios arquitectónicos" establecieron el número de cubículos que habrá dentro de cada laboratorio, así que basta con determinar donde debe estar cada laboratorio, la ubicación de los

cubículos no se consideró.

Nos indicaron que los siguientes ocho espacios deben permanecer en la planta baja por lo pesado de su equipo o por conveniencia, por lo que no se consideran parte del problema.

- Laboratorio de Manufactura Avanzada
- Laboratorio de Manufactura Convencional
- Laboratorio de Procesamiento de Plásticos
- UDIATEM (Unidad de Investigación y Asistencia Técnica en Materiales) Pesado
- Proyectos (en el que se exhibe el proyecto ganador)
- Exhibición
- Laboratorio de Pruebas Mecánicas
- Laboratorio de Doble Altura

Los espacios que se están tomando en consideración son los siguientes:

	Espacio	Ubicación en los planos	Área en m^2 según archivo de "Requerimientos de espacios"	Área en m^2 del espacio en los planos
1	Biomecánica	PB	10×15=150	10×16.3=163
2	Metalografía	PB	10×15=150	10×17.9=179
3	LIMAC 1	1er piso	10×15=150	10×17.6=176
4	LIMAC 2	1er piso	10×15=150	10×17.6=176
5	LIMAC 3	1er piso	10×15=150	10×17.6=176
6	Coordinación de Cómputo	1er piso	10×15=150	10×16=160
7	Área flexible para Proyectos 2	1er piso	10×15=150	10×17.6=176
8	Métodos, Ergonomía y Logística	1er piso	10×15=150	10×16=160
9	Ingeniería Industrial y Depto. I. I.	1er piso	10×15=150	10×17.6=176
10	Proyectos Mecatrónicos	2o. piso	10×15=150	10×17.6=176
11	Robótica y Mecanismos	2o. piso	10×7.5=75 10×7.5=75	10×15.7=157
12	Automatización Industrial	2o. piso	10×15=150	10×17.6=176

Espacio		Ubicación en los planos	Área en m^2 según archivo de "Requerimientos de espacios"	Área en m^2 del espacio en los planos
13	Lab. de Ing. Médica y Depto. I. M.	2o. piso	10×15=150	10×17.6=176
14	Lab. de Ing. Médica	2o. piso	10×15=150	10×16=160
15	Aula 1	PB	10×7.5=75	10×5=50
16	Jefatura DIMEI	1er. piso	10×15=150	10×5=50
17	Sala de juntas DIMEI	1er. piso	10×7.5=75	10×4.8=48
18	Área secretarias	1er. piso	Incluida en 16	10×5=50
19	Aula 3	PB	10×7.5=75	5×8.4=42.4
20	Aula 4	PB	10×7.5=75	5×8.4=42.4
21	Metalografía y pruebas mecánicas inv.	1er. piso	10×15=150	10×4.2=42.9
22	Microscopia/Caracterización	1er. piso	10×7.5=75	10×4.2=42.9
23	Área de trabajo de mecatrónica	2o. piso	Incluida en Depto. de Mecatrónica	6.5×6.4=41.9
24	Aula 2	PB	10×7.5=75	5×7.2=36.4
25	Corrosión	1er. piso	10×7.5=75	4×8.2=33
26	Área de proyectos UDIATEM	1er. piso	10×15=150	13.2×15.6=205.9
27	Proy. de soc. estudiantiles	1er.piso	10×15=150	10×20=200
28	Aula 5	1er.piso	10×7.5=75	10×12.7=127.8
29	Aula 6	1er.piso	10×7.5=75	10×12.7=127.8
30	Sala de videoconf. y control	1er. piso	10×15=150	10×9.7=97
31	Aula 8	2o. piso	10×7.5=75	10×9.2=92.7
32	Sala de profesores	1er.piso	10×7.5=75	7.7×7.6=59.1
33	Aula 7	2o.piso	10×7.5=75	7.5×8.2=62.3
34	Depto. de manufactura y materiales	2o. piso	3 módulos de 10×15=150	22.8×20=456.4
35	Depto. de Ing. de diseño CDMIT	2o. piso	3 módulos de 10×15=150	22.7×20=454.5
36	Diseño mecatrónico y Departamento de mecatrónica	2o. piso (3 mod.)	10×15=150 10×15=150	17.6×20=352.8
37	Área flexible para Proyectos 3	2o. piso	10×15=150	17.6×20=352.4
38	Análisis experimental	1er. piso	10×15=150	4×4.7=18.9

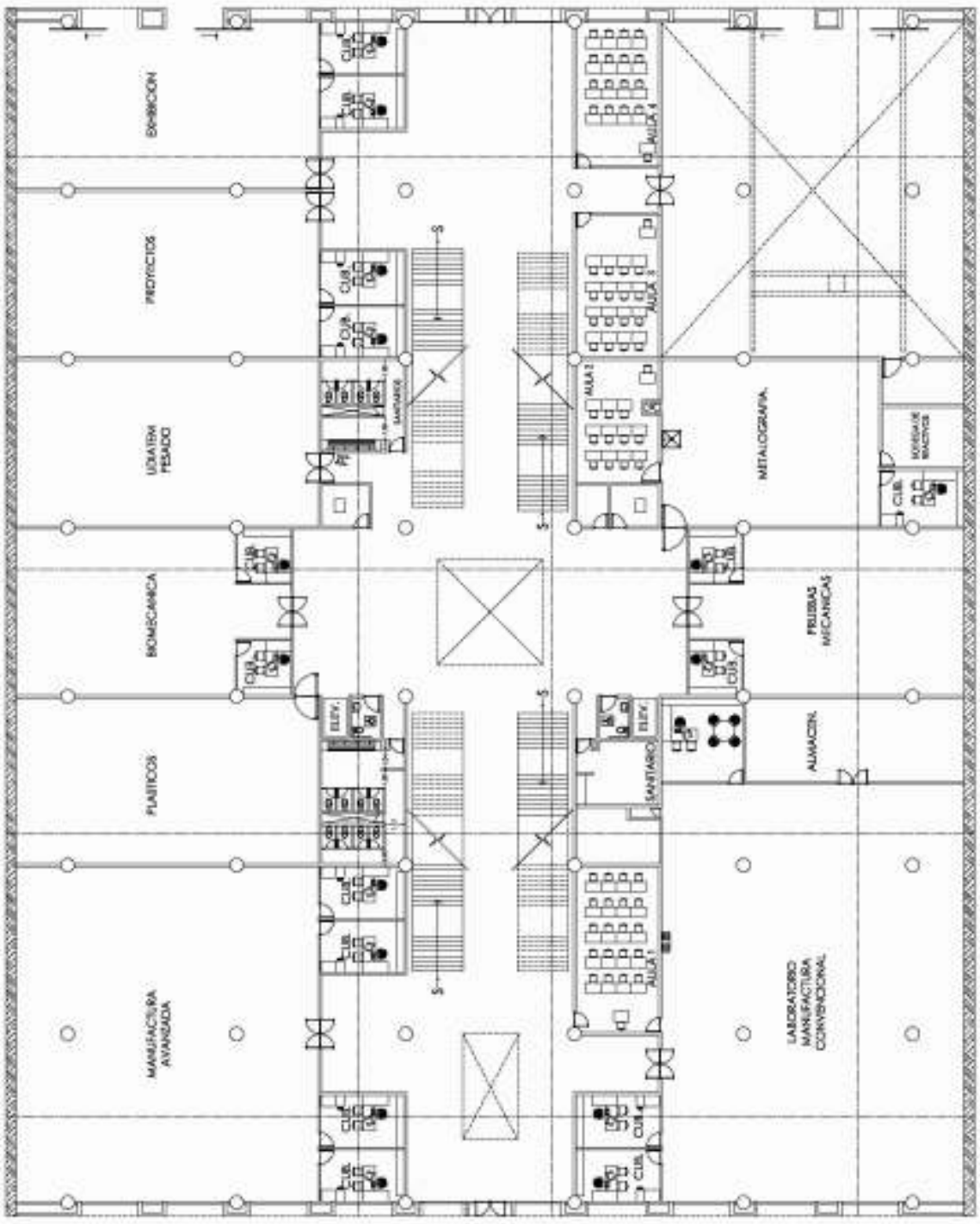


Figura 4.1: Planta Baja del Centro de Ingeniería Avanzada

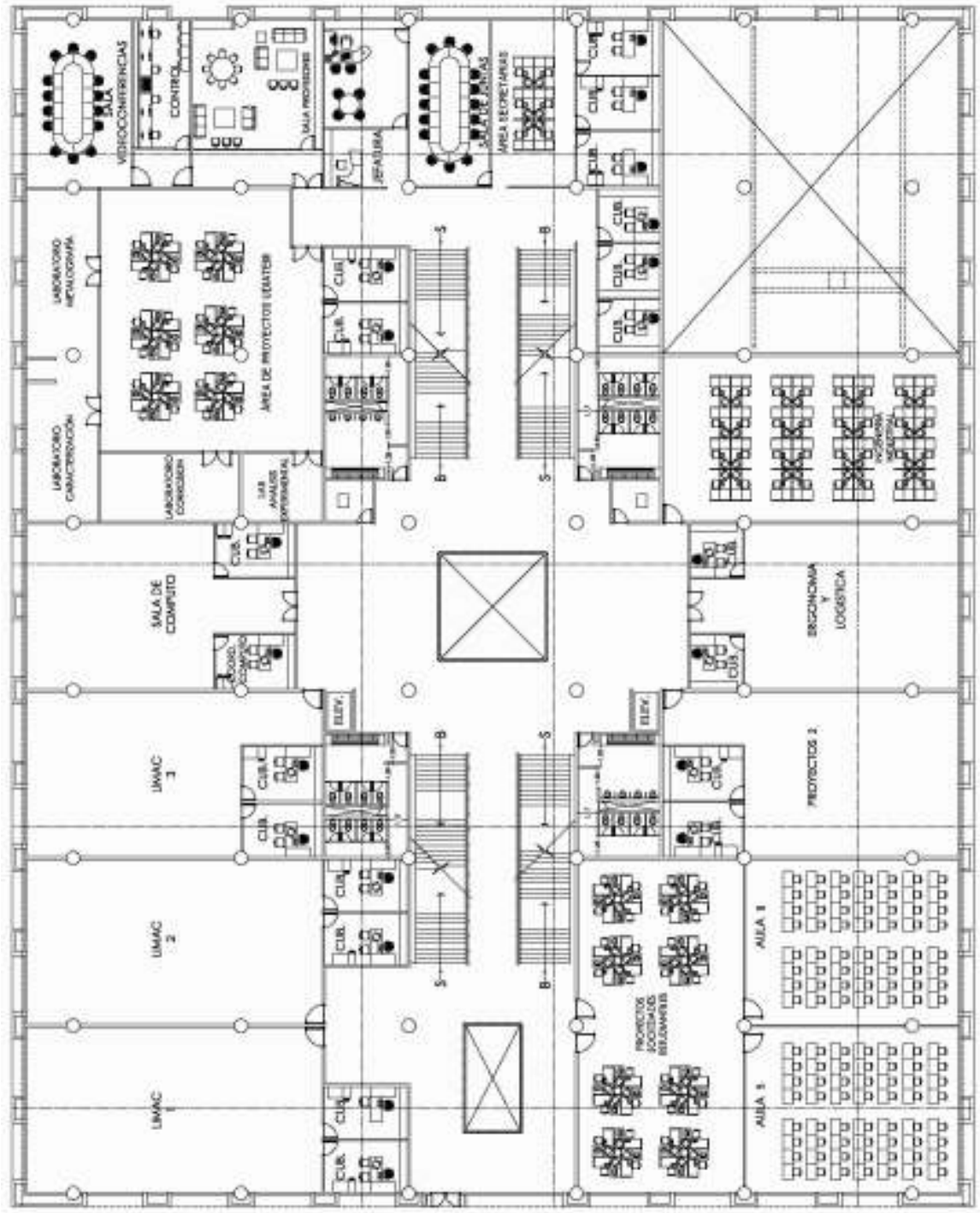


Figura 4.2: Primer Piso del Centro de Ingeniería Avanzada

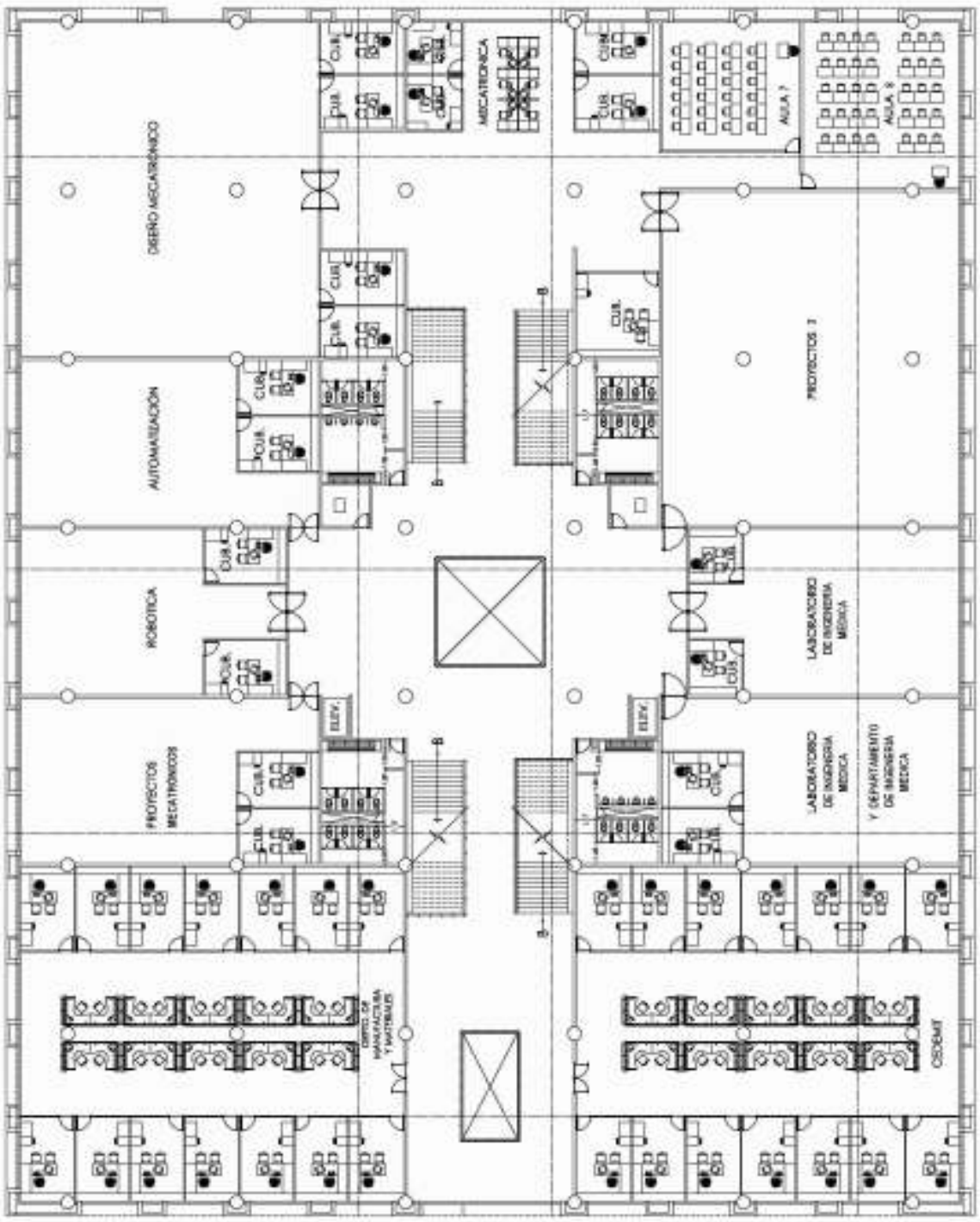


Figura 4.3: Segundo Piso del Centro de Ingeniería Avanzada

Existe un importante flujo de personas que toman clase en el edificio y se van, para que el modelo refleje esta situación se consideró a los dos entradas como espacios ficticios, por lo que el tamaño del problema a resolver es 40. A las entradas principal (en la planta baja a un lado de Exhibición, ver figura 4.1) y secundaria se les asignaron los números 39 y 40 respectivamente.

El algoritmo de Búsqueda Dispersa que se implementó para resolver este Problema de Asignación Cuadrática requiere como datos de entrada de dos matrices cuadradas del tamaño del problema (40): una de flujos y otra de distancias. Cada entrada de la matriz de flujo F_{ij} indica el número de usuarios por semana que se mueven del espacio i al espacio j. Cada entrada de la matriz de distancias D_{ij} representa la distancia en decímetros de la entrada del espacio i a la entrada del espacio j. En ambos casos i y j toman los 40 valores de los espacios antes mencionados.

Para la determinación de la matriz de flujos me proporcionaron los horarios de las materias que se imparten en el edificio "Alberto Camacho" y acceso a la lista de alumnos de cada curso, aunque me advirtieron que casi no había movimiento entre aulas. Se determinó el tamaño de muestra para una proporción con la fórmula que se presenta a continuación:

$$n = \frac{N}{\varepsilon^2(N-1)+1}$$

Donde N es el número de grupos y ε el error. Con un error del 10 % había que revisar 69 de 214 grupos. Comencé a revisar, pero debido al cambio de semestre perdí el acceso a las listas de alumnos, traté junto con las personas involucradas en el proyecto de recuperar el acceso a dichas listas. Hasta ese momento había revisado 5 de los 69 grupos, la revisión consistía en determinar si el alumno que estaba tomando un curso a una cierta hora un cierto día de la semana venía de tomar un curso en la hora anterior o iba a tomar un curso en la hora que le seguía, revisar esos cinco grupos implicó la revisión de 62 listas de alumnos. En lo que había revisado se observó que la matriz de flujo no es simétrica, también se observó que el movimiento entre aulas era prácticamente nulo, lo que coincidía con lo que se me dijo al inicio, por lo que con mi tutor se decidió no revisar el movimiento entre aulas, considerarlo cero salvo para aquellos espacios en los que ya se había determinado cual era el movimiento. Había muchos espacios para los que no podía determinar el flujo, por ello las personas que trabajan en el proyecto de construcción del Centro de Ingeniería Avanzada me proporcionaron la mayor parte de la matriz de flujo. Asimismo, ellos me informaron que estiman que el 70 % de

las personas accederán al edificio por la entrada principal (la que está en la planta baja a un lado de Exhibición, ver figura 4.1) y el 30 % restante por la entrada secundaria.

Para la mayoría de los espacios del nuevo Centro de Ingeniería Avanzada se tiene una estimación del número de usuarios por semana, ya sea porque dicha estimación se estableció en los cuestionarios por los encargados de los laboratorios o porque las personas involucradas en el proyecto me la proporcionaron. Sin embargo, había cinco salones para los que no se tenía una estimación del número de usuarios por semana. Las personas encargadas del proyecto me permitieron repartir las clases de los salones AC01, AC02 y (8 de las 11 clases del salón) 418 en los tres salones de la planta baja (1, 3 y 4) y dos del primer piso. Se impartieron 38 cursos en esos salones en el segundo semestre de 2010. Como los salones del primer piso son muy grandes, se ordenaron los salones por número de alumnos y asignamos los 14 grupos más grandes a los salones del primer piso (7 cursos a cada salón) y 8 cursos a cada uno de los tres salones sin ocupar de la planta baja. Con los horarios de los cursos se determino el número de usuarios por semana de estos espacios. La persona designada para atenderme me informó que en al Aula 2 se impartirán las clases de Metalografía.

Para la determinación de la matriz de distancias, la distancia a considerar entre espacios debe ser la más corta. Esta matriz es simétrica, para determinarla se tuvieron que calcular 780 distancias entre espacios. Las matrices de distancias y flujos del problema en cuestión se presentan en las figuras 4.4 y 4.5, respectivamente.

El Problema de Asignación Cuadrática no considera restricciones de espacio en la asignación, esto es, considera que cualquier laboratorio puede ser asignado a cualquier espacio físico, no obstante en este caso se tienen restricciones de espacio, no se puede asignar Biomecánica al Aula 1 porque el área de Biomecánica es de $150m^2$ mientras que el área del Aula 1 es de $50m^2$. En la tabla anterior el área de los primeros 14 espacios está dentro del intervalo $[157, 179]$ metros cuadrados (aunque de acuerdo al archivo de "requerimientos de espacios" el área que requieren para cada uno de estos espacios es de $150m^2$), se consideró a estos espacios como intercambiables. También se consideraron como intercambiables los siguientes espacios:

Espacio	El área del espacio es o está en el intervalo
15, 16, 17, 18	[48,50] m^2

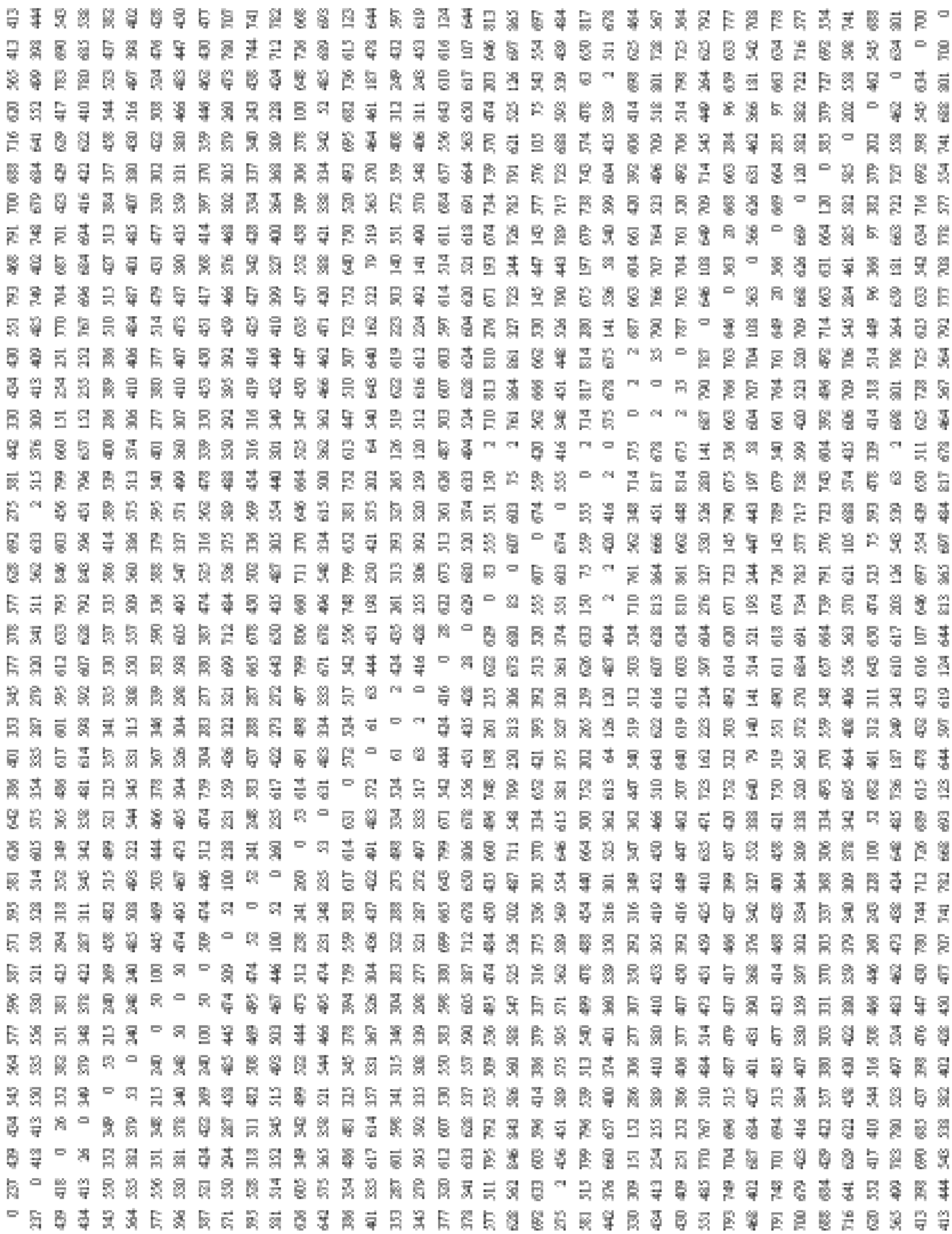


Figura 4.4: Matriz de distancias del CIA, cada distancia esta dada en decímetros lineales

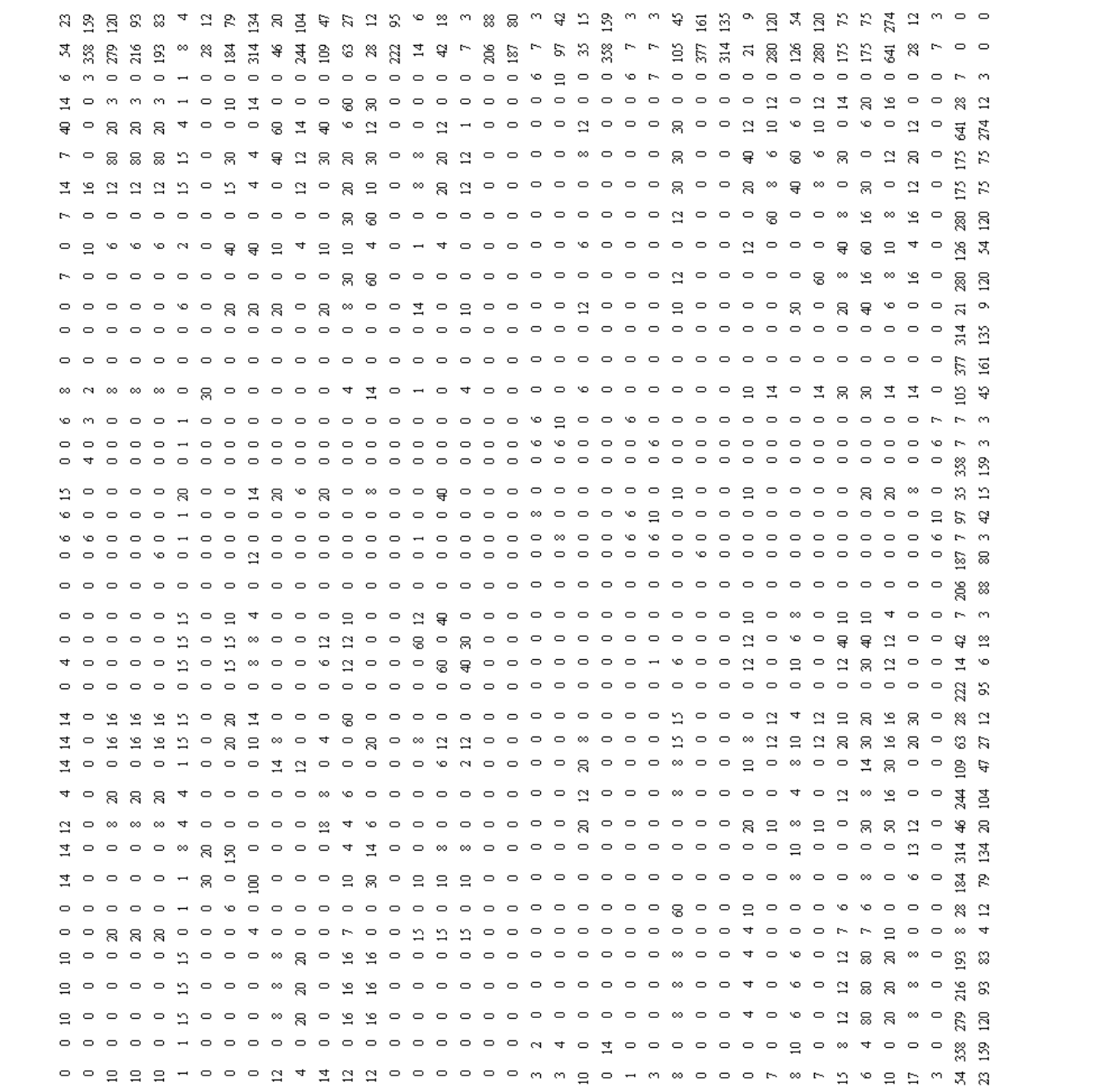


Figura 4.5: Matriz de flujos del CIA, cada entrada representa el número de usuarios por semana

Espacio	El área del espacio es o está en el intervalo
19, 20, 21, 22, 23	[41.9,42.9] m^2
24, 25	[33,36.4] m^2
26, 27	[200,205.92] m^2
28, 29	127.8 m^2
30, 31	[92.7,97] m^2
32, 33	[59.1,62.3] m^2
34, 35	[454.5,456.4] m^2
36, 37	[352.4,352.8] m^2

En los planos el área del espacio 38, Análisis experimental, es de 18.9 m^2, ningún otro espacio es tan pequeño, ese espacio va a quedar fijo. Las entradas ficticias sólo pueden ser asignadas a ellas mismas y se pueden intercambiar entre sí.

Se modificó el algoritmo de Búsqueda Dispersa implementado para que tomara en cuenta las restricciones. Se realizaron 400 corridas, el tiempo promedio de una corrida es de 25 segundos. El valor de la mejor solución encontrada es de 1,135,296.6 metros lineales por semana, la permutación que le corresponde es:

$$(2, 12, 7, 5, 3, 11, 4, 9, 8, 6, 13, 14, 1, 10, 15, 18, 17, 16, 19, 20, 22, 21,$$
$$23, 24, 25, 26, 27, 29, 28, 30, 31, 32, 33, 34, 35, 37, 36, 38, 40, 39)$$

La asignación de espacios que proporciona esta permutación es esta:

Espacio físico en los planos	Espacio que debe estar allí
Planta Baja	
Biomecánica	Metalografía
Metalografía	Automatización Industrial
Aula 1	Aula 1
Aula 2	Aula 2
Aula 3	Aula 3
Aula 4	Aula 4
Primer Piso	
LIMAC 1	Área flexible para Proyectos 2
LIMAC 2	LIMAC 3

Espacio físico en los planos	Espacio que debe estar allí
Primer Piso	
LIMAC 3	LIMAC 1
Coordinación de Cómputo	Robótica y Mecanismos
Área flexible para Proyectos 2	LIMAC 2
Métodos, Ergonomía y Logística	Ing. Ind. y Depto de Ing. Ind.
Ing. Ind. y Depto de Ing. Ind.	Métodos, Ergonomía y Logística
Jefatura DIMEI	Área secretarias
Sala de juntas DIMEI	Sala de juntas DIMEI
Área secretarias	Jefatura DIMEI
Metalografía y pruebas mec. inv.	Microscopia/Caracterización
Microscopia/Caracterización	Metalografía y pruebas mec. inv.
Corrosión	Corrosión
Área de proyectos UDIATEM	Área de proyectos UDIATEM
Proy. de soc. estudiantiles	Proy. de soc. estudiantiles
Aula 5	Aula 6
Aula 6	Aula 5
Sala de videoconf. y control	Sala de videoconf. y control
Sala de profesores	Sala de profesores
Análisis Experimental	Análisis Experimental
Segundo Piso	
Proyectos Mecatrónicos	Coordinación de Computo
Robótica y Mecanismos	Lab. de Ing. Médica y Depto. I. M.
Automatización Industrial	Lab de Ing. Médica
Lab. de Ing. Médica y Depto. I. M.	Biomecánica
Lab de Ing. Médica	Proyectos Mecatrónicos
Área de trabajo de mecatrónica	Área de trabajo de mecatrónica
Aula 7	Aula 7
Aula 8	Aula 8
Depto. de manufactura y mat.	Depto. de manufactura y mat.
Depto de Ing. de diseño CDMIT	Depto de Ing. de diseño CDMIT
Diseño mecatrónico y Depto. de M.	Área flexible para Proyectos 3
Área flexible para Proyectos 3	Diseño mecatrónico y Depto. de M.

En las figuras 4.6, 4.7 y 4.8 se muestra la asignación de espacios de la mejor

solución encontrada. Los espacios cuyo nombre esta subrayado no estaban originalmente en donde aparecen aquí.

En la solución encontrada la entrada principal (en la planta baja a un lado de Exhibición) y la entrada secundaria deben intercambiar sus lugares para que la distancia que se recorra dentro del Centro de Ingeniería Avanzada (CIA) sea la mejor que se encontró 1,135,296.6 metros lineales por semana. Si las entradas se quedan en sus lugares respectivos la distancia que se recorre dentro del CIA es de 1,144,268.8 metros lineales por semana, es decir, el valor de la función objetivo se incrementa en 8,972.2 metros lineales por semana, lo que representa menos de un 1 % de incremento.

Se comentó la solución encontrada con las personas involucradas en el proyecto. En la solución presentada se llegó a lo siguiente:

Espacio físico en los planos	Espacio que debe estar allí
Primer Piso	
LIMAC 1	Área flexible para Proyectos 2
Área flexible para Proyectos 2	LIMAC 2

Sin embargo, el LIMAC 2 es un laboratorio que requiere de muchas computadoras y por cuestiones de cableado ellos consideran más adecuado que el LIMAC 2 quede en el espacio físico del LIMAC 1 y que el Área flexible para Proyectos 2 se quede en su lugar. Si se realiza este cambio en la mejor solución que se encontró la distancia a recorrer dentro del CIA sería de 1,143,058.1 metros lineales por semana, lo que representa un incremento en el valor de la función objetivo de 7,761.5 metros lineales por semana, es decir, menos de un 1 % de incremento.

Si cada espacio se deja en donde estableció el arquitecto se recorrerían 1,206,399.7 metros lineales por semana, esto es, las personas caminarían 71,103.1 metros lineales más por semana, los cuales representan un incremento del 6 % en la distancia recorrida.

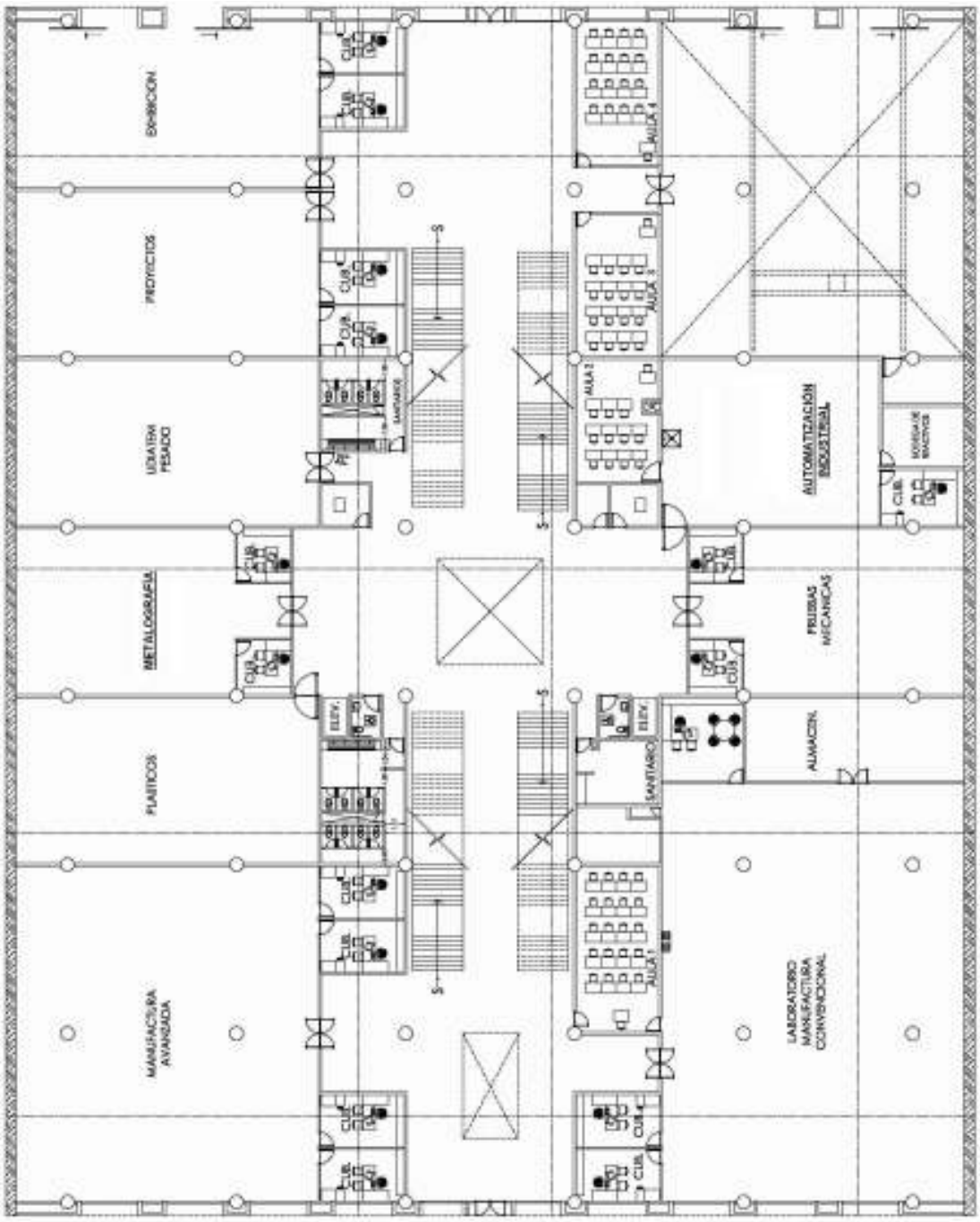


Figura 4.6: Planta baja del CIA de acuerdo a la asignación de espacios de la mejor solución encontrada

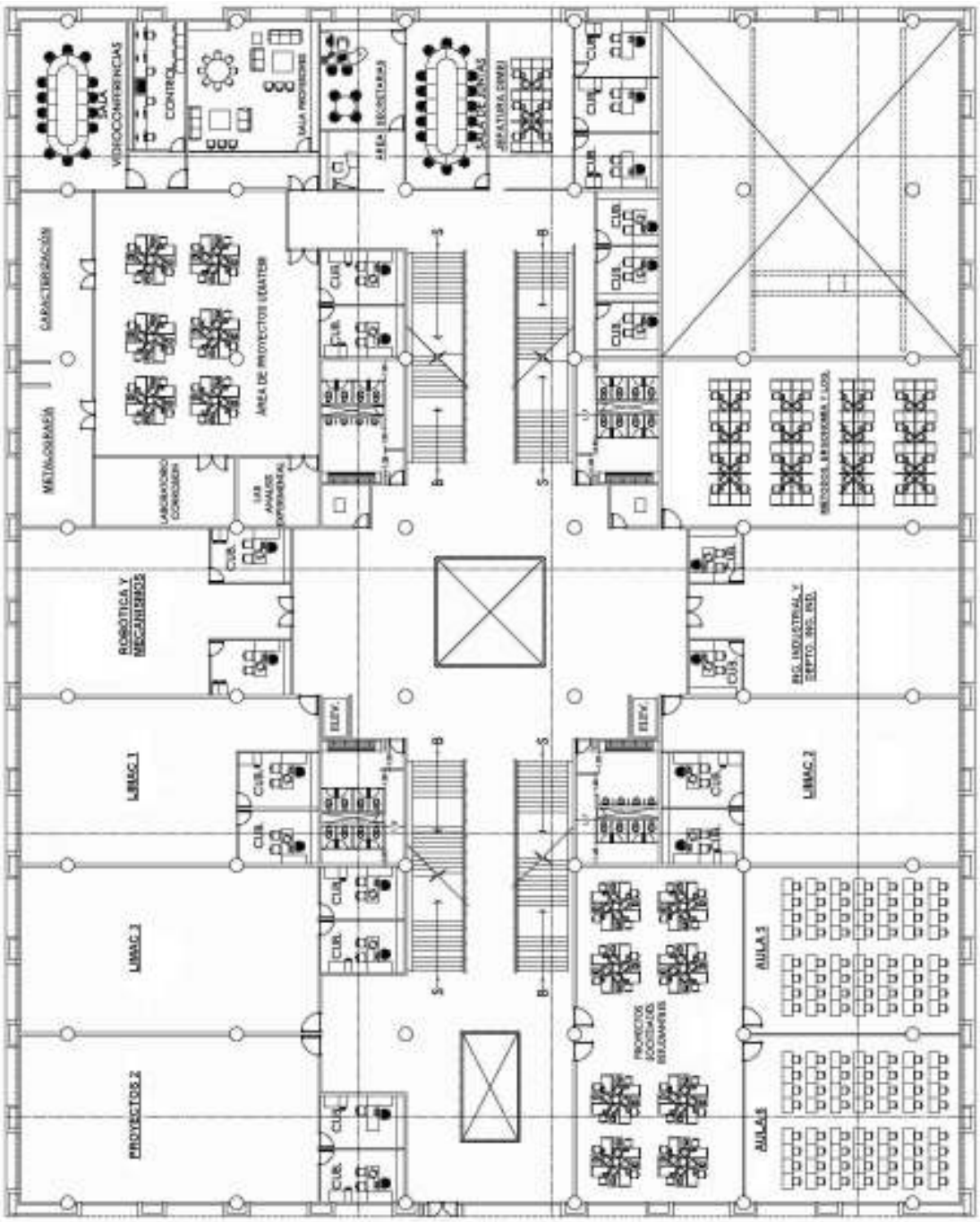


Figura 4.7: Primer piso del CIA de acuerdo a la asignación de espacios de la mejor solución encontrada

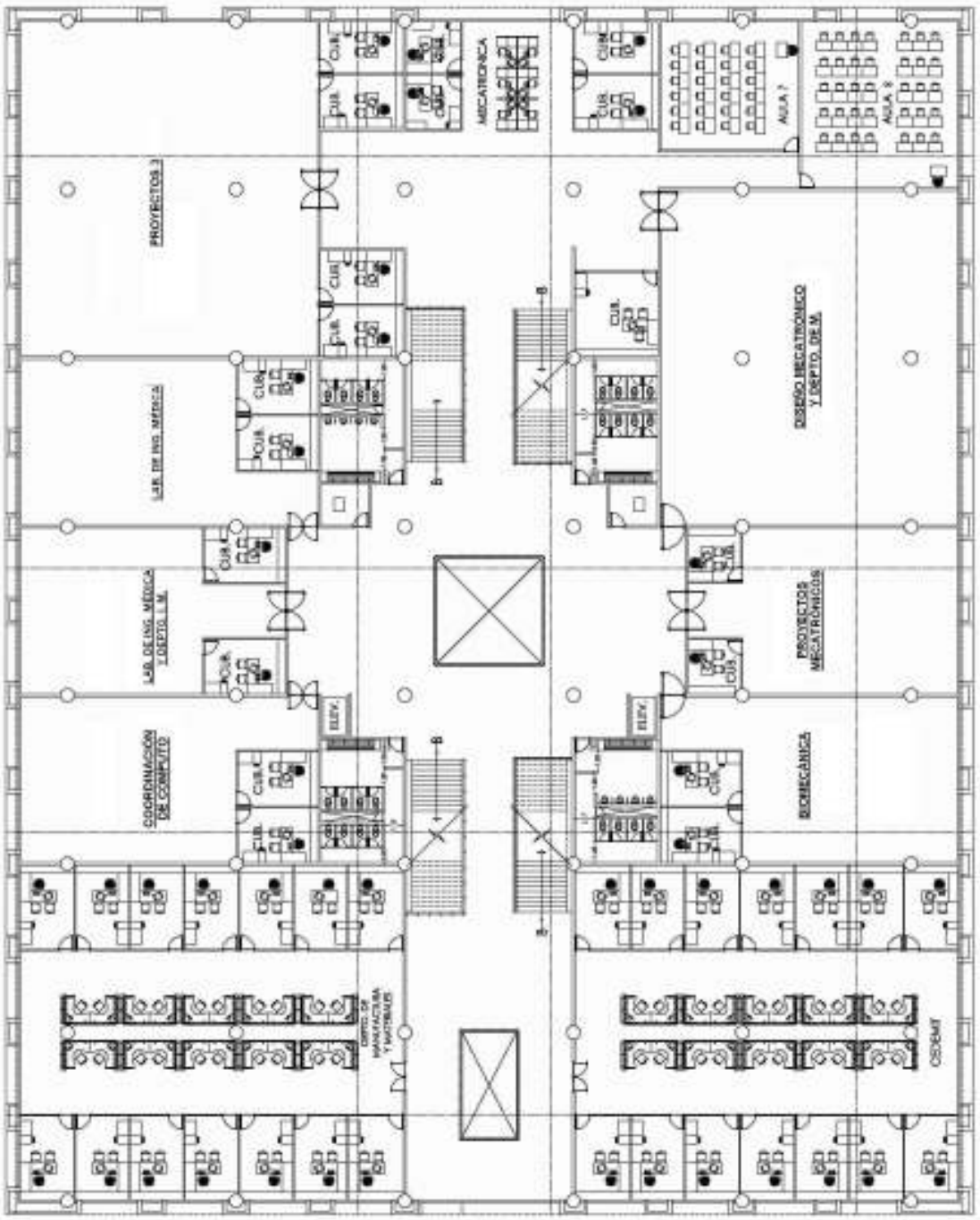


Figura 4.8: Segundo piso del CIA de acuerdo a la asignación de espacios de la mejor solución encontrada

Capítulo 5

Conclusiones

La selección de parámetros es difícil, se trabajó bastante en ello, sin embargo, considero que los resultados del algoritmo que se presento en el Capítulo 3 pueden mejorarse si se efectúa una mejor selección de los parámetros v (vecinos a revisar) y m (número de veces que se repite el procedimiento mejora) que se emplean en el procedimiento "mejora".

Asimismo, habría que trabajar más con el valor del parámetro $iteSinM$ (iteraciones sin que el conjunto referencia, E se modifique) del procedimiento de Búsqueda dispersa del Capítulo 3, para mejorar el tiempo de corrida del algoritmo. Ver que tanto puede reducirse este parámetro, permitiendo al mismo tiempo que el número de iteraciones que se realizan logre alcanzar el óptimo.

En el algoritmo heurístico de Búsqueda Dispersa que se implementó fue posible considerar las restricciones del caso real y encontrar una solución al mismo.

La asignación de espacios que de acuerdo a los planos del edificio puede darse tiene tantas restricciones que deja muy poca libertad de acción. Los planos del edificio no son acordes a los "requerimientos de espacios" que me proporcionaron las personas involucradas en el proyecto de construcción del Centro de Ingeniería Avanzada, de acuerdo a la información de los "requerimientos de espacios" requieren de 24 espacios son de 150 m^2 y 14 de 75 m^2. Sin embargo, en los planos del edificio hay 14 espacios de 150 m^2, 14 espacios de más o menos de 50 m^2, 4 de más o menos 100 m^2, 2 de 200 m^2, 2 de 350 m^2 y 2 de 450 m^2. Considero que si los planos fueran más acordes con los requerimientos, la solución podría haber sido muy diferente de la que se obtuvo.

El tiempo empleado para resolver una instancia no es una función lineal de la velocidad del procesador. La velocidad del procesador de la computadora en el que se realizaron las primeras corridas era de 1.7 GHz, después se uso una computadora cuya velocidad del procesador era de 2.4 GHz, la velocidad del procesador se incremento en un 41 %, sin embargo, el porcentaje en que se redujo el tiempo de corrida no es del 41 %.

Apéndice A

Algoritmo

Se presenta aquí el programa en Delphi 5 de un algoritmo de búsqueda dispersa para el Problema de Asignación Cuadrática.

```
unit UnitBD13;
interface
uses
 Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
 Dialogs, StdCtrls;
const
 R=5; {Número máximo de soluciones a combinar}
type
 matriz = array[1..150,1..150] of Integer;
 matriz01 = array[1..150,1..150] of 0..1;
 permutacion = array[1..150] of 1..150;
 sol = record
        P:permutacion;
        MP:matriz01;
        valor:Integer;
       end;
 coleccion = array[1..150] of permutacion;
 elite = array[1..20] of sol;
 pareja=record
         v,p:Integer;
        end;
 permuta=array[1..150] of pareja;
```

```
 string255 = string[255];
 conjunto=set of 0..255;
 TForm1 = class(TForm)
   Button1: TButton;
   Button2: TButton;
   Button3: TButton;
   Button4: TButton;
   Button5: TButton;
   Button6: TButton;
   Button7: TButton;
   Button8: TButton;
   Button9: TButton;
   Button10: TButton;
   Button12: TButton;
   procedure FormClick(Sender: TObject);
   procedure Button1Click(Sender: TObject);
   procedure Button2Click(Sender: TObject);
   procedure Button3Click(Sender: TObject);
   procedure Button4Click(Sender: TObject);
   procedure Button5Click(Sender: TObject);
   procedure Button6Click(Sender: TObject);
   procedure Button7Click(Sender: TObject);
   procedure Button8Click(Sender: TObject);
   procedure Button9Click(Sender: TObject);
   procedure Button10Click(Sender: TObject);
   procedure Button12Click(Sender: TObject);
 private
   { Private declarations }
 public
   { Public declarations }
 end;
var
 A,B:matriz;
 E:elite;
 archi:TextFile;
 Form1: TForm1;
implementation
{$R *.DFM}
```

```
procedure TForm1.FormClick(Sender: TObject);
begin
 ShowMessage('Por favor, pulse algún botón');
end;

function distCuadrada(A,B:permutacion;const n:integer):Integer;
{* Esta función de distancia es equivalente a la función de *}
{* distancia euclideana entre A y B que aparece en la sec-  *}
{* ción II del artículo de Cung.                            *}
 var
  i,d:Integer;
 begin
  d:=0;
  for i:=1 to n do
   if A[i]<>B[i] then d:=d+2;
  distCuadrada:=d;
 end;

procedure generaSolAle(var X:permutacion;const n:integer);
{* Este procedimiento genera una permutación aleatoria X *}
var
 orden:conjunto;
 i,j:Integer;
begin
 Randomize;
 orden:=[1..n];
 i:=1;
 Repeat
  j:=trunc(random*n)+1;
  if j in orden then
   begin
    X[i]:=j;
    exclude(orden,j);
    i:=i+1;
   end;
 until i=n+1;
end;
```

```
procedure genera_n_1_sol(var C:coleccion;const n:integer);
{* Este procedimiento genera n-1 permutaciones circulares *}
{* equivalentes a la permutación C[1].                    *}
var
 i,j:Integer;
begin
 for i:=2 to n do
  begin
  for j:=1 to n-1 do
    C[i][j]:=C[i-1][j+1];
   C[i][n]:=C[i-1][1];
  end;
end;

procedure leeMatrices(var A,B:matriz;const n:integer;
                      const t:string255);
var
 arch:TextFile;
 i,j:integer;
begin
 AssignFile(arch,t);
 Reset(arch);
 if n>=19 then
  begin
   for i:=1 to n do
    for j:=1 to n do
     read(arch,A[i,j]);
   for i:=1 to n do
    for j:=1 to n do
     read(arch,B[i,j]);
  end;
 CloseFile(arch);
end;

function evaluaPer(A,B:matriz;Q:permutacion;
                   const n:integer):Integer;
{* Esta función calcula el valor de la función *}
{*objetivo en la permutación Q                 *}
```

```
var
 j,k,v:Integer;
begin
 v:=0;
 for j:=1 to n do
  for k:=1 to n do
   v:=v+A[j,k]*B[Q[j],Q[k]];
 evaluaPer:=v;
end;

procedure ordena(var C:elite;const desde,hasta:Integer);
{* Este procedimiento ordena los elementos del arreglo C *}
{* de menor a mayor con respecto al valor de C[i].valor  *}
var
 Y:sol;
 i,k:Integer;
 encontrado:Boolean;
begin
 for k:=desde to hasta do
  begin
   Y:=C[k];
   i:=k-1;
   encontrado:=false;
   while (i>=1) and (not encontrado) do
    if Y.valor<C[i].valor then
     begin
      C[i+1]:=C[i];
      i:=i-1
     end
    else
     encontrado:=true;
   C[i+1]:=Y
  end;
end;

procedure mejora(const n,rep:integer;var E:elite;var cambio:boolean);
type
 pareja=array[1..2] of 0..150;
```

```
 conj=array[1..250] of pareja;
var
 i,j,k,v,m,indi,replicas,ve,temporal:integer;
 act,menor:sol;
 matrizCero:matriz01;
 conjPer:conj;
 esta:boolean;
begin
 for i:=1 to n do
  for j:=1 to n do
   matrizCero[i,j]:=0;
 Randomize;
 case n of
        19: replicas:=14;
    20..26: replicas:=16;
    27..50: replicas:=30;
    51..89: replicas:=50;
   90..100: replicas:=250;
  101..150: replicas:=400
 end;
 case n of
   19..20: ve:=80;
   21..90: ve:=200;
  91..150: ve:=250;
 end;
 if rep>0 then replicas:=rep;
 temporal:=E[1].valor;
 for v:=1 to replicas do
  begin
   if n<20 then
    begin
     for i:=1 to n do
      begin
       m:=1;
       menor:=E[i];
       repeat
        act:=E[i];
        repeat
```

```
      j:=trunc(random*(n-1))+1;
      repeat
       k:=trunc(random*n)+1;
      until j<k;
      conjPer[m][1]:=j;
      conjPer[m][2]:=k;
      indi:=1;
      esta:=false;
      while (indi<m) and (not esta) do
       if (conjPer[m][1]=conjPer[indi][1]) and
          (conjPer[m][2]=conjPer[indi][2]) then
         esta:=true
       else
         indi:=indi+1;
     until (not esta);
     act.P[j]:=E[i].P[k];
     act.P[k]:=E[i].P[j];
     act.valor:=evaluaPer(A,B,act.P,n);
     if act.valor<menor.valor then menor:=act;
     m:=m+1;
    until m>ve;
    if menor.valor<E[i].valor then
     begin
      E[i]:=menor;
      E[i].MP:=matrizCero;
      for j:=1 to n do E[i].MP[E[i].P[j],j]:=1
     end;
   end; {for i:=1 to n ... }
  ordena(E,2,n);
 end {if n<20 then...}
else
 begin
  for i:=1 to 20 do
   begin
    m:=1;
    menor:=E[i];
    repeat
     act:=E[i];
```

```
          repeat
           j:=trunc(random*(n-1))+1;
           repeat
            k:=trunc(random*n)+1;
           until j<k;
           conjPer[m][1]:=j;
           conjPer[m][2]:=k;
           indi:=1;
           esta:=false;
           while (indi<m) and (not esta) do
            if (conjPer[m][1]=conjPer[indi][1]) and
               (conjPer[m][2]=conjPer[indi][2]) then
              esta:=true
            else
              indi:=indi+1;
          until (not esta);
          act.P[j]:=E[i].P[k];
          act.P[k]:=E[i].P[j];
          act.valor:=evaluaPer(A,B,act.P,n);
          if act.valor<menor.valor then menor:=act;
          m:=m+1;
         until m>ve;
         if menor.valor<E[i].valor then
          begin
           E[i]:=menor;
           E[i].MP:=matrizCero;
           for j:=1 to n do E[i].MP[E[i].P[j],j]:=1
          end;
       end; {for i:=1 to 20 ... }
      ordena(E,2,20);
     end; {else}
  end; {for v:=1 to ...}
  if E[1].valor<temporal then cambio:=true;
end; {procedure mejora}

procedure operadorP(const n,ite:integer;var E:elite;
                    const P:permutacion;var menor:sol);
{* Recibe la permutación P, revisa 80 (ó 200 ó ite)  *}
```

```
{* vecinos distintos de esta. Deja en "menor" a P, o *}
{* al vecino que tiene el menor valor de la función. *}
type
 pareja=array[1..2] of 0..150;
 conj=array[1..253] of pareja;
var
 j,k,m,indi,ochenta200:integer;
 act:sol;
 conjPer:conj;
 esta:boolean;
begin
 Randomize;
 if n<=20 then
  ochenta200:=80
 else
  ochenta200:=200;
 if ite>0 then ochenta200:=ite;
 m:=1;
 menor.P:=P;
 menor.valor:=evaluaPer(A,B,P,n);
 repeat
  act.P:=P;
  repeat
   j:=trunc(random*(n-1))+1;
   repeat
    k:=trunc(random*n)+1;
   until j<k;
   conjPer[m][1]:=j;
   conjPer[m][2]:=k;
   indi:=1;
   esta:=false;
   while (indi<m) and (not esta) do
    if (conjPer[m][1]=conjPer[indi][1]) and
       (conjPer[m][2]=conjPer[indi][2]) then
      esta:=true
    else
      indi:=indi+1;
  until (not esta);
```

```
  act.P[j]:=P[k];
  act.P[k]:=P[j];
  act.valor:=evaluaPer(A,B,act.P,n);
  if act.valor<menor.valor then menor:=act;
  m:=m+1;
 until m>ochenta200;
end; {procedure operadorP}

procedure pobIni(var E:elite;const n:integer);
var
 C:coleccion;
 i,j,k,d:Integer;
 matrizCero:matriz01;
 menor:sol;
 tempo:permutacion;
 cambio:boolean;
begin
 generaSolAle(C[1],n);
 genera_n_1_sol(C,n);
 tempo:=C[1];
 if n<=20 then
  begin
   for i:=1 to n do operadorP(n,0,E,C[i],E[i]);
   ordena(E,2,n);
   repeat
    generaSolAle(C[1],n);
    d:=distCuadrada(C[1],tempo,n);
   until (d>0) and (d<2*n);
   genera_n_1_sol(C,n);
   for i:=1 to n do
    begin
     operadorP(n,0,E,C[i],menor);
     if menor.valor<E[n].valor then
      begin
       E[n]:=menor;
       ordena(E,n,n);
      end;
    end;
```

```
  for i:=1 to n do
   for j:=1 to n do
    matrizCero[i,j]:=0;
  for k:=1 to n do
   begin
    E[k].MP:=matrizCero;
    for j:=1 to n do E[k].MP[E[k].P[j],j]:=1;
   end
 end {begin if n<=20}
else
 begin
  for i:=1 to 20 do operadorP(n,0,E,C[i],E[i]);
  ordena(E,2,20);
  for i:=21 to n do
   begin
    operadorP(n,0,E,C[i],menor);
    if menor.valor<E[20].valor then
     begin
      E[20]:=menor;
      ordena(E,20,20);
     end;
   end;
  repeat
   generaSolAle(C[1],n);
   d:=distCuadrada(C[1],tempo,n);
  until (d>0) and (d<2*n);
  genera_n_1_sol(C,n);
  for i:=1 to n do
   begin
    operadorP(n,0,E,C[i],menor);
    if menor.valor<E[20].valor then
     begin
      E[20]:=menor;
      ordena(E,20,20);
     end;
   end;
  for i:=1 to n do
   for j:=1 to n do
```

```
      matrizCero[i,j]:=0;
    for k:=1 to 20 do
     begin
      E[k].MP:=matrizCero;
      for j:=1 to n do E[k].MP[E[k].P[j],j]:=1;
     end;
  end; {else}
  mejora(n,0,E,cambio);
  writeln(archi,'PobIni ');
  if n<20 then
     for k:=1 to n do writeln(archi,IntToStr(E[k].valor))
  else
     for k:=1 to 20 do writeln(archi,IntToStr(E[k].valor));
end;

function suma(A:matriz;B:matriz01;const n:integer):matriz;
var
 i,j:Integer;
 sum:matriz;
begin
 for i:=1 to n do
  for j:=1 to n do
   sum[i,j]:=A[i,j]+B[i,j];
 suma:=sum;
end;

function sumaMat(A:matriz01;B:matriz01;const
                 n:integer):matriz;
var
 i,j:Integer;
 sum:matriz;
begin
 for i:=1 to n do
  for j:=1 to n do
   sum[i,j]:=A[i,j]+B[i,j];
 sumaMat:=sum;
end;
```

```
procedure ordenaPer(var Z:permuta;const n:integer);
{* Este procedimiento recibe en Z un renglón de una   *}
{* matriz de nXn y ordena las entradas del renglón de *}
{* menor a mayor, por ejemplo, si recibe como entrada *}
{* (0,1,2,1) devuelve (0,1,1,2).                      *}
var
 Y:pareja;
 i,k:Integer;
 encontrado:Boolean;
begin
 for k:=2 to n do
  begin
   Y.v:=Z[k].v;
   Y.p:=Z[k].p;
   i:=k-1;
   encontrado:=false;
   while (i>=1) and (not encontrado) do
    if Y.v<Z[i].v then
     begin
      Z[i+1]:=Z[i];
      i:=i-1
     end
    else
     encontrado:=true;
   Z[i+1].v:=Y.v;
   Z[i+1].p:=Y.p;
  end
end;

procedure cambiarSiMejoraAlPeor(var E:elite;const n,v:Integer;
   const P:permutacion; const s:string255;var cambio:boolean);
var
 i,j:Integer;
 matrizCero:matriz01;
begin
 for i:=1 to n do
  for j:=1 to n do
   matrizCero[i,j]:=0;
```

```
 if n<=20 then
  begin
   if v<E[n].valor then
    begin
     writeln(archi,s,IntToStr(v),' < ',IntToStr(E[n].valor),
              '=E[',IntToStr(n),']');
     cambio:=true;
     E[n].valor:=v;
     E[n].P:=P;
     E[n].MP:=matrizCero;
     for j:=1 to n do E[n].MP[E[n].P[j],j]:=1;
     ordena(E,n,n);
    end;
  end
 else
  if v<E[20].valor then
   begin
    writeln(archi,s,IntToStr(v),' < ',IntToStr(E[20].valor),
             ' =E[20]');
    cambio:=true;
    E[20].valor:=v;
    E[20].P:=P;
    E[20].MP:=matrizCero;
    for j:=1 to n do E[20].MP[E[20].P[j],j]:=1;
    ordena(E,20,20);
   end;
end;

procedure combinaSol(var E:elite;var F:matriz; const n:
                     integer;var P:permutacion; var v1:
                     integer;var X:matriz01;var sitios,
                     edificios:conjunto;var cambio:boolean);
(* combinaSol=combina soluciones *)
(* Se elige aleatoriamente un número j en {2,3,4,5} de    *)
(* acuerdo a una distribución uniforme. Nuevamente, sigui-*)
(* endo una distribución uniforme se eligen j matrices de *)
(* permutación en E, se las suma y se guarda la suma en T.*)
(* Cada matriz de permutación que se elige en E se suma a *)
```

```
(* la matriz de frecuencias F, para que cada entrada de la*)
(* matriz F reporte el no. de veces que un edificio es    *)
(* asignado a un sitio, la matriz F se emplea en el proce-*)
(* dimiento de diversificación.                           *)
(* sitios={1,...,n}, edificios={1,...,n}, aleatoriamente, *)
(* siguiendo una distribución uniforme, se elige un sitio *)
(* en sitios, elegir un sitio equivale a elegir un renglón*)
(* de la matriz T, se ordenan las entradas de ese renglón *)
(* de menor a mayor y se guarda en un conjunto a todas las*)
(* que tienen el mayor valor y se elige aleatoriamente una*)
(* de ellas, siguiendo una distribución uniforme. De esta *)
(* manera se obtiene una solución factible X (matriz de   *)
(* permutación) a partir de T. Se determina la permutación*)
(* correspondiente a X, se aplica el operadorP a dicha    *)
(* permutación. Se guarda la solución mejorada en P y su  *)
(* valor en v1. Se modifica E, si es necesario.           *)
var
 ordenE,eleg:conjunto;
 i,j,k:Integer;
 Z:permuta; {* X es la matriz correspondiente a la solución *}
 Q:sol;     {* (permutación) que genera este procedimiento. *}
 T:matriz;
 matrizCero:matriz01;
 s:string;
begin
 for i:=1 to n do
  for j:=1 to n do
   begin
    T[i,j]:=0;
    MatrizCero[i,j]:=0;
   end;
 Randomize;
 {*** Inicia obtención de una solución combinada T ***}
 repeat   {*R=5 es el número máximo de soluciones a combinar*}
  j:=trunc(random*R)+1;
 until j>1;
 {* Se eligen j matrices de permutaciones y *}
 {* se suman, la suma se guarda en T.       *}
```

```
if n<=20 then
 begin
  ordenE:=[1..n];
  k:=trunc(random*n)+1;
  F:=suma(F,E[k].MP,n);
  T:=suma(T,E[k].MP,n);
  for i:=2 to j do
   begin
    exclude(ordenE,k);
    repeat
     k:=trunc(random*n)+1;
    until k in ordenE;
    F:=suma(F,E[k].MP,n);
    T:=suma(T,E[k].MP,n);
   end;
 end {begin if n<=...}
else
 begin
  ordenE:=[1..20];
  k:=trunc(random*20)+1;
  F:=suma(F,E[k].MP,n);
  T:=suma(T,E[k].MP,n);
  for i:=2 to j do
   begin
    exclude(ordenE,k);
    repeat
     k:=trunc(random*20)+1;
    until k in ordenE;
    F:=suma(F,E[k].MP,n);
    T:=suma(T,E[k].MP,n);
   end;
 end; {begin else}
{*** Termina obtención de una solución combinada T  ***}
{*** Inicia construcción de una solución factible X ***}
{*** a partir de T.                                 ***}
repeat
 repeat
  j:=trunc(random*n)+1;
```

```
  until j in sitios;
  exclude(sitios,j);
  for i:=1 to n do
   begin
    Z[i].v:=T[j,i];
    Z[i].p:=i
   end;
  ordenaPer(Z,n);
  k:=n;
  while not(Z[k].p in edificios) do k:=k-1;
  eleg:=[];
  for i:=1 to n do
   if (Z[k].v=Z[i].v) and (Z[i].p in edificios) then
      include(eleg,Z[i].p);
  repeat
   i:=trunc(random*n)+1
  until i in eleg;
  X[j,i]:=1;
  exclude(edificios,i);
 until sitios=[];
 F:=suma(F,X,n);
 {*** Los tres siguientes renglones transforman la matriz de ***}
 {*** permutación X en la permutación que le correponde, P   ***}
 for i:=1 to n do
  for j:=1 to n do
   if X[i,j]=1 then P[i]:=j;
 {*** Termina construcción de una solución factible X ***}
 {*** Aplicación del operador a la solución obtenida  ***}
 operadorP(n,0,E,P,Q);
 P:=Q.P;
 v1:=Q.valor;
 cambiarSiMejoraAlPeor(E,n,Q.valor,Q.P,'10 CombSol ',cambio);
end; {end procedure combinaSol}

procedure intensificacion(var E:elite;var F:matriz;  const n:
          integer;var v1:integer; var P:permutacion;const cs:
          boolean;var cambio:boolean);
{* En la sección III del art. de Cung establecen que un movi-*}
```

```
{* miento consiste en intercambiar los edificios localizados *}
{* sitios i y j, que la vecindad de un punto se define como  *}
{* el conjunto de soluciones que pueden ser alcanzadas en un *}
{* movimiento desde el punto, y que cada vez que un moviento *}
{* es realizado, el movimiento opuesto es tabú por un número *}
{* S de iteraciones.                                         *}
{* El número de vecinos de un punto son las combinaciones de *}
{* 2 en n, y son menos de 1200 para n<50. Para n<50 se creo  *}
{* este procedimiento que lo que hace es tomar un punto efec-*}
{* tuar un movimiento, cambiarse a ese punto, realizar un mo-*}
{* vimiento,y así se llevan a cabo 1200 iteraciones, se veri-*}
{* fica en cada movimiento que los últimos 8 movimientos     *}
{* sean distintos,es decir, S=8. En cada movimiento se compa-*}
{* ra la solución obtenida con el peor elemento de la pobla- *}
{* ción. La nueva solución es incluida en la población si es *}
{* mejor que el peor elemento.                               *}
{* La figura I del art. de Cung indica que se realiza inten- *}
{* sificación después de generar una solución combinada. Con *}
{* instancias de distintos tamaños revise que resultaba      *}
{* mejor. Para instancias menores a 27 conviene aplicar in-  *}
{* tensificación partiendo de la solución combinada que se   *}
{* acaba de generar, no obstante, para instancias entre 27 y *}
{* 49 conviene partir de la mejor solución que se tiene. Este*}
{* procedimiento considera lo que resulta mejor, pero la in- *}
{* dicación de esto aparece en el procedimiento busquedaD.   *}
type
 lista=array[0..7] of permutacion;
var
 L:lista;
 i,j,k,m,q,indi,tope,v2:integer;
 X:matriz01;
 sitios,edificios:conjunto;
 esta:boolean;
begin
 for i:=1 to n do
  for j:=1 to n do
   X[i,j]:=0;
 sitios:=[1..n];
```

```
edificios:=[1..n];
if cs then combinaSol(E,F,n,P,v1,X,sitios,edificios,cambio);
tope:=1200;
L[0]:=P;
for m:=1 to 7 do
 begin
  repeat
   j:=trunc(random*n)+1;
   repeat
    k:=trunc(random*n)+1
   until j<>k;
   L[m]:=L[m-1];
   L[m][j]:=L[m-1][k];
   L[m][k]:=L[m-1][j];
   indi:=0;
   esta:=false;
   while (m>1) and (indi<m) and (not esta) do
    begin
     q:=1;
     esta:=(L[m][q]=L[indi][q]);
     while esta and (q<n) do
      begin
       q:=q+1;
       esta:=(L[m][q]=L[indi][q])
      end;
     if (not esta) then indi:=indi+1;
    end; {while (m>1) and (indi<m) and ...}
  until (not esta);
  v2:=evaluaPer(A,B,L[m],n);
  cambiarSiMejoraAlPeor(E,n,v2,L[m],'20 intensi ',cambio);
 end; {for m:=1 to 7 ...}
m:=7;
esta:=false;
for i:=8 to tope do
  begin
   repeat
    if (not esta) then m:=(m+1) mod 8;
    j:=trunc(random*n)+1;
```

```
        repeat
         k:=trunc(random*n)+1
        until j<>k;
        if m=0 then
         begin
          L[0]:=L[7];
          L[0][j]:=L[7][k];
          L[0][k]:=L[7][j]
         end
        else
         begin
          L[m]:=L[m-1];
          L[m][j]:=L[m-1][k];
          L[m][k]:=L[m-1][j]
         end;
        esta:=false;
        indi:=0;
        repeat
         while (not esta) and (indi<>m) and (indi<8) do
          begin
           q:=1;
           esta:=(L[m][q]=L[indi][q]);
           while esta and (q<n) do
            begin
             q:=q+1;
             esta:=(L[m][q]=L[indi][q])
            end;
           if (not esta) then indi:=indi+1;
          end; {while (not esta) ...}
         indi:=indi+1;
        until (indi>7) or esta;
       until (not esta);
       v2:=evaluaPer(A,B,L[m],n);
       cambiarSiMejoraAlPeor(E,n,v2,L[m],'20 intensi ',cambio);
      end {for i:=8 to ...}
    end; {procedure intensificacion...}
```

```
procedure intensificacion2(var E:elite;var F:matriz;const n:integer;
               var v1:Integer;var P:permutacion;var cambio:boolean);
{* Este procedimiento revisa 1200 vecinos si n>=50           *}
{* La figura I del art. de Cung indica que se realiza inten- *}
{* sificación después de generar una solución combinada, sin *}
{* embargo, observe que se obtienen mejores resultados par-  *}
{* tiendo de la mejor solución conocida, eso es lo que hace  *}
{* este procedimiento. Se guarda en la variable “menor” a la *}
{* permutación y el valor de la función objetivo en esa per- *}
{* mutación más pequeño que se encontró al revisar los veci- *}
{* nos correspondientes. Después de revisar a todos los ve-  *}
{* cinos, se compara el valor de “menor” con el del peor ele-*}
{* mento en la población, si el valor de "menor" es mejor que*}
{* el del peor elemento, “menor” se incluye en la población. *}
type
 pareja=array[1..2] of 0..150;
 listaTabu=array[0..199] of pareja;
var
 i,j,k,m,indi:Integer;
 act,menor:sol;
 esta:boolean;
 X:matriz01;
 sitios,edificios:conjunto;
 L:listaTabu;
begin
 for i:=1 to n do
  for j:=1 to n do
   X[i,j]:=0;
 sitios:=[1..n];
 edificios:=[1..n];
 if n>=50 then
  begin
   menor.P:=P;
   menor.valor:=v1;
   Randomize;
   for m:=0 to 199 do
    begin
     act.P:=P;
```

```
  repeat
   j:=trunc(random*(n-1))+1;
   repeat
    k:=trunc(random*n)+1;
   until j<k;
   L[m][1]:=j;
   L[m][2]:=k;
   indi:=0;
   esta:=false;
   while (indi<m) and (not esta) do
    if (L[m][1]=L[indi][1]) and (L[m][2]=L[indi][2]) then
     esta:=true
    else
     indi:=indi+1;
  until (not esta);
  act.P[j]:=P[k];
  act.P[k]:=P[j];
  act.valor:=evaluaPer(A,B,act.P,n);
  if act.valor<menor.valor then menor:=act;
 end; {for m:=0 ...}
m:=199;
esta:=false;
for i:=200 to 1199 do
 begin
  act.P:=P;
  repeat
   if (not esta) then m:=(m+1) mod 200;
   j:=trunc(random*(n-1))+1;
   repeat
    k:=trunc(random*n)+1;
   until j<k;
   L[m][1]:=j;
   L[m][2]:=k;
   indi:=0;
   esta:=false;
   repeat
    while (not esta) and (indi<>m) and (indi<200) do
     begin
```

```
          if (L[m][1]=L[indi][1]) and (L[m][2]=L[indi][2]) then
           esta:=true
          else
           indi:=indi+1
         end;
        indi:=indi+1
       until esta or (indi>199);
      until (not esta);
      act.P[j]:=P[k];
      act.P[k]:=P[j];
      act.valor:=evaluaPer(A,B,act.P,n);
      if act.valor<menor.valor then menor:=act;
     end; {for i:=200 ...}
    cambiarSiMejoraAlPeor(E,n,menor.valor,menor.P,
                          '20 intensi ',cambio);
  end;
end;

procedure diversificacion(var E:elite;var F:matriz;const n:integer;
            var P:permutacion; var X:matriz01;var sitios,edificios:
            conjunto;var v1:integer;var cambio:boolean);
(* Para tomar en cuenta la diversidad en la población se *)
(* asigna 0.05n de edificios a sitios que se han usado   *)
(* poco, para ello se emplea la matriz de frecuencias F  *)
(* (que se mencionó en el procedimiento "combinaSol") la *)
(* cual reporta en cada entrada el número de veces que un*)
(* edificio es asignado a un sitio. Las asignaciones res-*)
(* tantes, n-0.5n, se determinan con el procedimiento    *)
(* "combinaSol". El procedimiento que se sigue para de-  *)
(* terminar las 0.05n edificios es el este:              *)
(* sitios={1,...,n}, edificios={1,...,n}, aleatoriamente,*)
(* siguiendo una distribución uniforme, se elige un sitio*)
(* en sitios, elegir un sitio equivale a elegir un ren-  *)
(* glón de la matriz F, se ordenan las entradas de ese   *)
(* renglón de menor a mayor y se guarda en un conjunto a *)
(* todas las que tienen el menor valor y se elige aleato-*)
(* riamente una de ellas, siguiendo una distribución uni-*)
(* forme. De esta manera se obtienen una parte de la so- *)
```

```
(* lución factible X (matriz de permutación) el resto de *)
(* X se determina con el procedimiento “combinaSol”.     *)
var
 i,j,k,l,m:integer;
 Z:permuta;
 eleg:conjunto;
 matrizCero:matriz01;
 Q,U:sol;
 T:matriz;
begin
 m:=round(0.05*n);
 sitios:=[1..n];
 edificios:=[1..n];
 for i:=1 to n do
  for j:=1 to n do
   begin
    X[i,j]:=0;
    matrizCero[i,j]:=0
   end;
 for l:=1 to m do
  begin
   repeat
    j:=trunc(random*n)+1;
   until j in sitios;
   exclude(sitios,j);
   for i:=1 to n do
    begin
     Z[i].v:=F[j,i];
     Z[i].p:=i
    end;
   ordenaPer(Z,n);
   k:=1;
   while not(Z[k].p in edificios) do k:=k+1;
   eleg:=[];
   for i:=1 to n do
    if (Z[k].v=Z[i].v) and (Z[i].p in edificios) then
      include(eleg,Z[i].p);
   repeat
```

```
  i:=trunc(random*n)+1
 until i in eleg;
 X[j,i]:=1;
 exclude(edificios,i);
end; {end for l:=1...}
combinaSol(E,F,n,P,v1,X,sitios,edificios,cambio);
Q.valor:=v1;
Q.P:=P;
Q.MP:=matrizCero;
for j:=1 to n do Q.MP[Q.P[j],j]:=1;
if n<20 then
   m:=n
else
   m:=20;
for l:=1 to m do
 begin
  F:=suma(F,Q.MP,n);
  T:=sumaMat(Q.MP,E[l].MP,n);
  sitios:=[1..n];
  edificios:=[1..n];
  {*** Inicia construcción de una solución ***}
  {*** factible X a partir de T            ***}
  X:=matrizCero;
  repeat
   repeat
    j:=trunc(random*n)+1;
   until j in sitios;
   exclude(sitios,j);
   for i:=1 to n do
    begin
     Z[i].v:=T[j,i];
     Z[i].p:=i
    end;
   ordenaPer(Z,n);
   k:=n;
   while not(Z[k].p in edificios) do k:=k-1;
   eleg:=[];
   for i:=1 to n do
```

```
       if (Z[k].v=Z[i].v) and (Z[i].p in edificios) then
         include(eleg,Z[i].p);
      repeat
       i:=trunc(random*n)+1
      until i in eleg;
      X[j,i]:=1;
      exclude(edificios,i);
     until sitios=[];
     F:=suma(F,X,n);
     {*** Los tres siguientes renglones transforman la  ***}
     {*** matriz de permutación X en la permutación que ***}
     {*** le correponde, P.                             ***}
     for i:=1 to n do
      for j:=1 to n do
       if X[i,j]=1 then P[j]:=i;
     {*** Termina construcción de una solución factible  ***}
     {*** X a partir de T.                               ***}
     {*** Aplicación del operador a la solución obtenida ***}
     operadorP(n,0,E,P,U);
     P:=U.P;
     v1:=U.valor;
     cambiarSiMejoraAlPeor(E,n,U.valor,U.P,
                     '10 CombSol ',cambio);
   end;
end;

procedure busquedaD(const n:integer;const archivo:string255;
                    var E:elite);
(* Este es el programa principal, llama a algunos de los *)
(* procedimientos que lo preceden. Se realizan siete ite-*)
(* raciones ordinarias ("combinaSol"), una iteración de  *)
(* intensificación y una iteración de diversificación, se*)
(* repite este proceso hasta que el número de iteraciones*)
(* sin modificacion (IteSinM) es igual a 23 o el número  *)
(* de iteraciones llega a 500. Además este procedimiento *)
(* crea  un  archivo  "mejora.txt" en el que registra su *)
(* corrida.                                              *)
var
```

```
 i,j,k,v1,prom,mejor,corridas,iteSinM,promIte,mayor:Integer;
 F:matriz; {F matriz de frecuencias}
 P:permutacion;
 s:string;
 X,matrizCero:matriz01;
 sitios,edificios:conjunto;
 hora:TDateTime;
 cambio:boolean;
begin
 AssignFile(archi,'C:\carmen\tesisMaestria\progDelphi\mejora.txt');
 Rewrite(archi);
 leeMatrices(A,B,n,archivo);
 corridas:=2;
 hora:=Time;
 Form1.Caption:=TimeToStr(hora);
 writeln(archi,TimeToStr(hora));
 prom:=0;
 promIte:=0;
 for k:=1 to corridas do
 begin
  pobIni(E,n);
  for i:=1 to n do
   for j:=1 to n do
    begin
     F[i,j]:=0;
     MatrizCero[i,j]:=0;
    end;
  cambio:=false;
  iteSinM:=0;
  i:=0;
  while (iteSinM<23) and (i<500) do
   begin
    i:=i+1;
    writeln(archi,'Iteracion ',IntToStr(i));
    if (i mod 9)=8 then
     begin
      case n of
        19..26:intensificacion(E,F,n,v1,P,true,cambio);
```

```
    27..49:intensificacion(E,F,n,E[1].valor,E[1].P,
                           false,cambio);
   50..150:intensificacion2(E,F,n,E[1].valor,E[1].P,
                            cambio)
  end;
 end
else
 if (i mod 9)=0 then
  begin
   writeln(archi,'Pob');
   if n<20 then
    for j:=1 to n do writeln(archi,IntToStr(E[j].valor))
   else
    for j:=1 to 20 do
     writeln(archi,IntToStr(E[j].valor));
   diversificacion(E,F,n,P,X,sitios,edificios,v1,cambio);
   if cambio then
    begin
     writeln(archi,'Pob');
     if n<20 then
      for j:=1 to n do
       writeln(archi,IntToStr(E[j].valor))
     else
      for j:=1 to 20 do
       writeln(archi,IntToStr(E[j].valor));
    end
  end
 else
  begin
   sitios:=[1..n];
   edificios:=[1..n];
   X:=matrizCero;
   combinaSol(E,F,n,P,v1,X,sitios,edificios,cambio);
   if n<90 then
    begin
     if (i mod 18)=1 then
      case n of
           19: mejora(n,10,E,cambio);
```

```
          20..89: mejora(n,0,E,cambio);
         end
       end
      else
       case n of
              90: if (i mod 27)=1 then mejora(n,45,E,cambio);
        91..100: if (i mod 36)=1 then mejora(n,43,E,cambio);
       101..150: if i=1 then mejora(n,0,E,cambio);
       end;
     end;
   writeln(archi,'E[1].valor= ',IntToStr(E[1].valor));
   if cambio then
    begin
     cambio:=false;
     iteSinM:=0;
    end
   else
    iteSinM:=iteSinM+1;
  end; {while...}
 hora:=Time;
 s:='';
 for j:=1 to n do s:=s+IntToStr(E[1].P[j])+' ';
 writeln(archi,TimeToStr(hora),'E[1].P=',s);
 writeln(archi,'iteSinM=',IntToStr(iteSinM));
 prom:=prom+E[1].valor;
 promIte:=promIte+i;
 if k=1 then
  begin
   mejor:=E[1].valor;
   mayor:=i;
  end
 else
  begin
   if E[1].valor<mejor then mejor:=E[1].valor;
   if i>mayor then mayor:=i
  end;
end; {for k:=1 to ...}
prom:=round(prom/corridas);
```

```
 promIte:=round(promIte/corridas);
 ShowMessage('Promedio='+IntToStr(prom)+' Mejor='+
             IntToStr(mejor)+' promIte='+IntToStr(promIte)+
             ' maxIte='+IntToStr(mayor)+' '+TimeToStr(hora));
 writeln(archi,'Promedio=',IntToStr(prom),' Mejor=',
         IntToStr(mejor),' promIte=',IntToStr(promIte),
         ' maxIte= ',IntToStr(mayor));
 CloseFile(archi);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
 busquedaD(26,'C:\carmen\tesisMaestria\progDelphi\bur26a.txt',E)
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
 busquedaD(49,'C:\carmen\tesisMaestria\progDelphi\sko49.txt',E)
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
 busquedaD(19,'C:\carmen\tesisMaestria\progDelphi\els19.txt',E)
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
 busquedaD(40,'C:\carmen\tesisMaestria\progDelphi\tho40.txt',E)
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
 busquedaD(28,'C:\carmen\tesisMaestria\progDelphi\nug28.txt',E)
end;

procedure TForm1.Button6Click(Sender: TObject);
begin
 busquedaD(35,'C:\carmen\tesisMaestria\progDelphi\tai35b.txt',E)
end;
```

```
procedure TForm1.Button7Click(Sender: TObject);
begin
 busquedaD(30,'C:\carmen\tesisMaestria\progDelphi\kra30b.txt',E)
end;

procedure TForm1.Button8Click(Sender: TObject);
begin
 busquedaD(32,'C:\carmen\tesisMaestria\progDelphi\kra32.txt',E)
end;

procedure TForm1.Button9Click(Sender: TObject);
begin
 busquedaD(35,'C:\carmen\tesisMaestria\progDelphi\tai35a.txt',E)
end;

procedure TForm1.Button10Click(Sender: TObject);
begin
 busquedaD(50,'C:\carmen\tesisMaestria\progDelphi\lipa50b.txt',E)
end;

procedure TForm1.Button12Click(Sender: TObject);
begin
 busquedaD(150,'C:\carmen\tesisMaestria\progDelphi\tho150.txt',E)
end;

end.
```

Apéndice B

Algoritmo con restricciones

Se enlista aquí el programa en Delphi 5 del algoritmo de búsqueda dispersa con el que se resolvió el problema de asignación cuadrática con restricciones del Centro de Ingeniería Avanzada.

```
unit unitBD18c;
{*** Este programa sólo puede aplicarse a la instancia de ***}
{*** tamaño 40 del Centro de Ingeniería Avanzada.         ***}

interface

uses
 Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
 Dialogs, StdCtrls;

const
 R=5; {Número máximo de soluciones a combinar}

type
 matriz = array[1..150,1..150] of Integer;
 matriz01 = array[1..150,1..150] of 0..1;
 permutacion = array[1..150] of 1..150;
 sol = record
        P:permutacion;
        MP:matriz01;
        valor:Integer;
       end;
```

```
 coleccion = array[1..150] of permutacion;
 elite = array[1..20] of sol;
 pareja=record
         v,p:Integer;
        end;
 permuta=array[1..150] of pareja;
 string255 = string[255];
 conjunto=set of 0..255;
 TForm1 = class(TForm)
   Button1: TButton;
   Button2: TButton;
   Button3: TButton;
   Button4: TButton;
   Button5: TButton;
   Button6: TButton;
   Button7: TButton;
   procedure FormClick(Sender: TObject);
   procedure Button1Click(Sender: TObject);
   procedure Button2Click(Sender: TObject);
   procedure Button3Click(Sender: TObject);
   procedure Button4Click(Sender: TObject);
   procedure Button5Click(Sender: TObject);
   procedure Button6Click(Sender: TObject);
   procedure Button7Click(Sender: TObject);
 private
   { Private declarations }
 public
   { Public declarations }
 end;

var
 A,B:matriz;
 E:elite;
 archi:TextFile;
 Form1: TForm1;

implementation
```

```
{$R *.DFM}

procedure TForm1.FormClick(Sender: TObject);
begin
 ShowMessage('Por favor, pulse algún botón');
end;

function distCuadrada(A,B:permutacion;const p,q:integer):Integer;
 var
  i,d:Integer;
 begin
  d:=0;
  for i:=p to q do
   if A[i]<>B[i] then d:=d+2;
  distCuadrada:=d;
 end;

procedure generaSolAle(var X:permutacion;const inicio,fin:integer);
var
 orden:conjunto;
 i,j,k:Integer;
begin
 Randomize;
 k:=fin-inicio+1;
 orden:=[inicio..fin];
 i:=inicio;
 Repeat
  j:=trunc(random*k)+inicio;
  if j in orden then
   begin
    X[i]:=j;
    exclude(orden,j);
    i:=i+1;
   end;
 until i=fin+1;
end;

procedure leeMatrices(var A,B:matriz;const n:integer;
```

```
                        const t:string255);
var
 arch:TextFile;
 i,j:integer;
begin
 AssignFile(arch,t);
 Reset(arch);
 if n>18 then
  begin
   for i:=1 to n do
    for j:=1 to n do
     read(arch,A[i,j]);
   for i:=1 to n do
    for j:=1 to n do
     read(arch,B[i,j]);
  end;
 CloseFile(arch);
end;

procedure leePermu(var P:permutacion;const n:integer;
                   const archivo:string255);
 var
  arch:TextFile;
  i:integer;
 begin
  AssignFile(arch,archivo);
  Reset(arch);
  for i:=1 to n do read(arch,P[i]);
  CloseFile(arch);
 end;

function evaluaPer(A,B:matriz;Q:permutacion;
                   const n:integer):Integer;
var
 j,k,v:Integer;
begin
 v:=0;
 for j:=1 to n do
```

```
  for k:=1 to n do
   v:=v+A[j,k]*B[Q[j],Q[k]];
 evaluaPer:=v;
end;

procedure ordena(var C:elite;const desde,hasta:Integer);
var
 Y:sol;
 i,k:Integer;
 encontrado:Boolean;
begin
 for k:=desde to hasta do
  begin
   Y:=C[k];
   i:=k-1;
   encontrado:=false;
   while (i>=1) and (not encontrado) do
    if Y.valor<C[i].valor then
     begin
      C[i+1]:=C[i];
      i:=i-1
     end
    else
     encontrado:=true;
   C[i+1]:=Y
  end;
end;

procedure mejora(const n,rep:integer;var E:elite;var cambio:boolean);
type
 pareja=array[1..2] of 0..150;
 conj=array[1..250] of pareja;
var
 i,j,k,v,m,indi,replicas,ve,temporal,tam,inicio:integer;
 act,menor:sol;
 matrizCero:matriz01;
 conjPer:conj;
 esta:boolean;
```

```
begin
 for i:=1 to n do
  for j:=1 to n do
   matrizCero[i,j]:=0;
 Randomize;
 case n of
    20..26: replicas:=16;
    27..50: replicas:=30;
    51..89: replicas:=50;
   90..100: replicas:=250;
  101..150: replicas:=400
 end;
 case n of
       20: ve:=80;
   21..90: ve:=115; {Vecinos del Cia40d con restricciones}
  91..150: ve:=250;
 end;
 if rep>0 then replicas:=rep;
 temporal:=E[1].valor;
 for v:=1 to replicas do
  begin
   if n>=20 then
    begin
     for i:=1 to 20 do
      begin
       m:=1;
       menor:=E[i];
       repeat
        act:=E[i];
        repeat
         repeat
          j:=trunc(random*(n-1))+1;
         until j<>38;
         case j of
          1..14:begin
                 tam:=14;
                 inicio:=1
                end;
```

```
   15..18:begin
           tam:=4;
           inicio:=15
          end;
   19..23:begin
           tam:=5;
           inicio:=19
          end;
   24,26,28,30,32,34,36,39:k:=j+1;
   25,27,29,31,33,35,37,40:k:=j-1
  end;
  if j in [1..23] then
   repeat
    k:=trunc(random*tam)+inicio;
   until k<>j;
  conjPer[m][1]:=j;
  conjPer[m][2]:=k;
  indi:=1;
  esta:=false;
  while (indi<m) and (not esta) do
   if ((conjPer[m][1]=conjPer[indi][1]) and
       (conjPer[m][2]=conjPer[indi][2])) or
      ((conjPer[m][1]=conjPer[indi][2]) and
       (conjPer[m][2]=conjPer[indi][1])) then
     esta:=true
   else
     indi:=indi+1;
 until (not esta);
 act.P[j]:=E[i].P[k];
 act.P[k]:=E[i].P[j];
 act.valor:=evaluaPer(A,B,act.P,n);
 if act.valor<menor.valor then menor:=act;
 m:=m+1;
until m>ve;
if menor.valor<E[i].valor then
 begin
  E[i]:=menor;
  E[i].MP:=matrizCero;
```

```
          for j:=1 to n do E[i].MP[E[i].P[j],j]:=1
         end;
       end; {for i:=1 to }
      ordena(E,2,20);
     end;
  end; {for v:=1 to ...}
  if E[1].valor<temporal then cambio:=true;
end; {procedure mejora}

procedure indices(var j,k:integer;const n:integer);
var
 tam,inicio:integer;
begin
 repeat
  j:=trunc(random*n)+1;
 until j<>38;
 case j of
  1..14:begin
         tam:=14;
         inicio:=1
        end;
  15..18:begin
          tam:=4;
          inicio:=15
         end;
  19..23:begin
          tam:=5;
          inicio:=19
         end;
  24,26,28,30,32,34,36,39:k:=j+1;
  25,27,29,31,33,35,37,40:k:=j-1
 end;
 if j in [1..23] then
   repeat
    k:=trunc(random*tam)+inicio;
   until k<>j;
end;
```

```
procedure operadorP(const n,ite:integer;var E:elite;const P:
                    permutacion;var menor:sol);
type
 pareja=array[1..2] of 0..150;
 conj=array[1..253] of pareja;
var
 j,k,m,indi:integer;
 act:sol;
 conjPer:conj;
 esta:boolean;
begin
 Randomize;
 m:=1;
 menor.P:=P;
 menor.valor:=evaluaPer(A,B,P,n);
 repeat
  act.P:=P;
  repeat
   indices(j,k,n);
   conjPer[m][1]:=j;
   conjPer[m][2]:=k;
   indi:=1;
   esta:=false;
   while (indi<m) and (not esta) do
    if ((conjPer[m][1]=conjPer[indi][1]) and
        (conjPer[m][2]=conjPer[indi][2])) or
       ((conjPer[m][1]=conjPer[indi][2]) and
        (conjPer[m][2]=conjPer[indi][1])) then
      esta:=true
    else
      indi:=indi+1;
  until (not esta);
  act.P[j]:=P[k];
  act.P[k]:=P[j];
  act.valor:=evaluaPer(A,B,act.P,n);
  if act.valor<menor.valor then menor:=act;
  m:=m+1;
 until m>115;
```

```
end; {procedure operadorP}

procedure pobIniUnaParte(var C:coleccion);
var
 i,j,d,d2,d3:integer;
 r1:real;
begin
 for i:=2 to 14 do
  begin
   for j:=1 to 13 do C[i][j]:=C[i-1][j+1];
   C[i][14]:=C[i-1][1];
  end;
 generaSolAle(C[1],15,18);
 for i:=2 to 4 do
  begin
   for j:=15 to 17 do C[i][j]:=C[i-1][j+1];
   C[i][18]:=C[i-1][15]
  end;
 repeat
  generaSolAle(C[5],15,18);
  d:=distCuadrada(C[1],C[5],15,18)
 until (d>0) and(d<8);
 for i:=6 to 8 do
  begin
   for j:=15 to 17 do C[i][j]:=C[i-1][j+1];
   C[i][18]:=C[i-1][15]
  end;
 repeat
  generaSolAle(C[9],15,18);
  d:=distCuadrada(C[1],C[9],15,18);
  d2:=distCuadrada(C[5],C[9],15,18);
 until (d>0) and (d<8) and (d2>0) and (d2<8);
 for i:=10 to 12 do
  begin
   for j:=15 to 17 do C[i][j]:=C[i-1][j+1];
   C[i][18]:=C[i-1][15]
  end;
 repeat
```

```
 generaSolAle(C[13],15,18);
 d:=distCuadrada(C[1],C[13],15,18);
 d2:=distCuadrada(C[5],C[13],15,18);
 d3:=distCuadrada(C[9],C[13],15,18);
until (d>0) and (d<8) and (d2>0) and
      (d2<8) and (d3>0) and (d3<8);
for j:=15 to 17 do C[14][j]:=C[13][j+1];
C[14][18]:=C[13][15];
generaSolAle(C[1],19,23);
for i:=2 to 5 do
 begin
  for j:=19 to 22 do C[i][j]:=C[i-1][j+1];
  C[i][23]:=C[i-1][19]
 end;
repeat
 generaSolAle(C[6],19,23);
 d:=distCuadrada(C[1],C[6],19,23);
until (d>0) and (d<10);
for i:=7 to 10 do
 begin
  for j:=19 to 22 do C[i][j]:=C[i-1][j+1];
  C[i][23]:=C[i-1][19]
 end;
repeat
 generaSolAle(C[11],19,23);
 d:=distCuadrada(C[1],C[11],19,23);
 d2:=distCuadrada(C[6],C[11],19,23);
until (d>0) and (d<10) and (d2>0) and (d2<10);
for i:=12 to 14 do
 begin
  for j:=19 to 22 do C[i][j]:=C[i-1][j+1];
  C[i][23]:=C[i-1][19]
 end;
randomize;
for i:=1 to 14 do
 begin
  for j:=12 to 18 do
   begin
```

```
      r1:=random;
      if r1<0.5 then
       begin
        C[i][2*j]:=2*j;
        C[i][2*j+1]:=2*j+1;
       end
      else
       begin
        C[i][2*j]:=2*j+1;
        C[i][2*j+1]:=2*j;
       end;
     end;
     C[i][38]:=38;
     r1:=random;
     if r1<=0.5 then
      begin
       C[i][39]:=39;
       C[i][40]:=40;
      end
     else
      begin
       C[i][39]:=40;
       C[i][40]:=39;
      end
   end;
end;

procedure pobIni(var E:elite;const n:integer);
var
 C:coleccion;
 i,j,k,d,d2,d3,d4,d5:Integer;
 matrizCero:matriz01;
 menor:sol;
 tempo,tempo2,tempo3,tempo4,tempo5:permutacion;
 cambio:boolean;
begin
 generaSolAle(C[1],1,14);
 pobIniUnaParte(C);
```

```
tempo:=C[1];
for i:=1 to 14 do operadorP(40,0,E,C[i],E[i]);
repeat
 generaSolAle(C[1],1,14);
 d:=distCuadrada(C[1],tempo,1,14)
until (d>0) and(d<28);
pobIniUnaParte(C);
for i:=1 to 6 do operadorP(40,0,E,C[i],E[i+14]);
ordena(E,2,20);
for i:=7 to 14 do
 begin
  operadorP(40,0,E,C[i],menor);
  if menor.valor<E[20].valor then
   begin
    E[20]:=menor;
    ordena(E,20,20);
   end;
 end;
tempo2:=C[1];
repeat
 generaSolAle(C[1],1,14);
 d:=distCuadrada(C[1],tempo,1,14);
 d2:=distCuadrada(C[1],tempo2,1,14)
until (d>0)and(d<28)and(d2>0)and(d2<28);
pobIniUnaParte(C);
for i:=1 to 14 do
 begin
  operadorP(40,0,E,C[i],menor);
  if menor.valor<E[20].valor then
   begin
    E[20]:=menor;
    ordena(E,20,20);
   end;
 end;
tempo3:=C[1];
repeat
 generaSolAle(C[1],1,14);
 d:=distCuadrada(C[1],tempo,1,14);
```

```
 d2:=distCuadrada(C[1],tempo2,1,14);
 d3:=distCuadrada(C[1],tempo3,1,14);
until (d>0)and(d<28)and(d2>0)and
      (d2<28)and(d3>0)and(d3<28);
pobIniUnaParte(C);
for i:=1 to 14 do
 begin
  operadorP(40,0,E,C[i],menor);
  if menor.valor<E[20].valor then
   begin
    E[20]:=menor;
    ordena(E,20,20);
   end;
 end;
tempo4:=C[1];
repeat
 generaSolAle(C[1],1,14);
 d:=distCuadrada(C[1],tempo,1,14);
 d2:=distCuadrada(C[1],tempo2,1,14);
 d3:=distCuadrada(C[1],tempo3,1,14);
 d4:=distCuadrada(C[1],tempo4,1,14);
until (d>0)and(d<28)and(d2>0)and(d2<28)and
      (d3>0)and(d3<28)and(d4>0)and(d4<28);
pobIniUnaParte(C);
for i:=1 to 14 do
 begin
  operadorP(40,0,E,C[i],menor);
  if menor.valor<E[20].valor then
   begin
    E[20]:=menor;
    ordena(E,20,20);
   end;
 end;
tempo5:=C[1];
repeat
 generaSolAle(C[1],1,14);
 d:=distCuadrada(C[1],tempo,1,14);
 d2:=distCuadrada(C[1],tempo2,1,14);
```

```
  d3:=distCuadrada(C[1],tempo3,1,14);
  d4:=distCuadrada(C[1],tempo4,1,14);
  d5:=distCuadrada(C[1],tempo5,1,14);
 until (d>0)and(d<28)and(d2>0)and(d2<28)and(d3>0)and
       (d3<28)and(d4>0)and(d4<28)and(d5>0)and(d5<28);
 pobIniUnaParte(C);
 for i:=1 to 14 do
  begin
   operadorP(40,0,E,C[i],menor);
   if menor.valor<E[20].valor then
    begin
     E[20]:=menor;
     ordena(E,20,20);
    end;
  end;
 writeln(archi,'PobIni');
 for i:=1 to n do
  for j:=1 to n do
   matrizCero[i,j]:=0;
 for k:=1 to 20 do
  begin
   E[k].MP:=matrizCero;
   for j:=1 to n do E[k].MP[E[k].P[j],j]:=1;
   writeln(archi,IntToStr(E[k].valor));
  end;
 mejora(n,0,E,cambio);
 writeln(archi,'PobIni mejorada');
 for k:=1 to 20 do writeln(archi,IntToStr(E[k].valor));
end;

function suma(A:matriz;B:matriz01;const n:integer):matriz;
var
 i,j:Integer;
 sum:matriz;
begin
 for i:=1 to n do
  for j:=1 to n do
   sum[i,j]:=A[i,j]+B[i,j];
```

```
 suma:=sum;
end;

function sumaMat(A:matriz01;B:matriz01;
                 const n:integer):matriz;
var
 i,j:Integer;
 sum:matriz;
begin
 for i:=1 to n do
  for j:=1 to n do
   sum[i,j]:=A[i,j]+B[i,j];
 sumaMat:=sum;
end;

procedure ordenaPer(var Z:permuta;const desde,hasta:integer);
var
 Y:pareja;
 i,k:Integer;
 encontrado:Boolean;
begin
 for k:=desde to hasta do
  begin
   Y.v:=Z[k].v;
   Y.p:=Z[k].p;
   i:=k-1;
   encontrado:=false;
   while (i>=1) and (not encontrado) do
    if Y.v<Z[i].v then
     begin
      Z[i+1]:=Z[i];
      i:=i-1
     end
    else
     encontrado:=true;
   Z[i+1].v:=Y.v;
   Z[i+1].p:=Y.p;
  end
```

```
 end;

procedure cambiarSiMejoraAlPeor(var E:elite;const n,v:Integer;const
               P:permutacion;const s:string255;var cambio:boolean);
var
 i,j:Integer;
 matrizCero:matriz01;
begin
 for i:=1 to n do
  for j:=1 to n do
   matrizCero[i,j]:=0;
 if (n>=20) and (v<E[20].valor) then
  begin
   writeln(archi,s,IntToStr(v),' < ',IntToStr(E[20].valor),
           ' =E[20]');
   cambio:=true;
   E[20].valor:=v;
   E[20].P:=P;
   E[20].MP:=matrizCero;
   for j:=1 to n do E[20].MP[E[20].P[j],j]:=1;
   ordena(E,20,20);
  end;
end;

procedure construSolFact(var sitios,edificios:conjunto;var j:
 integer;const de,a,m:integer;var X:matriz01;const T:matriz);
var
 i,k:integer;
 Z:permuta;
 eleg:conjunto;
begin
 repeat
  exclude(sitios,j);
  for i:=de to a do
   begin
    Z[i].v:=T[j,i];
    Z[i].p:=i
   end;
```

```
  ordenaPer(Z,de+1,a);
  k:=a;
  while not(Z[k].p in edificios) do k:=k-1;
  eleg:=[];
  for i:=de to a do
   if (Z[k].v=Z[i].v) and (Z[i].p in edificios) then
     include(eleg,Z[i].p);
  repeat
   i:=trunc(random*m)+de
  until i in eleg;
  X[j,i]:=1;
  exclude(edificios,i);
  if sitios<>[] then
    begin
     repeat
      j:=trunc(random*m)+de;
     until j in sitios;
    end
 until sitios=[];
end;

procedure construPermuFact(const T:matriz;var F:matriz;
          const n:integer;var sitios,edificios:conjunto;
          var X:matriz01;var P:permutacion;var v1:integer;
          var cambio:boolean;var E:elite);
var
 i,j,de,a,m:integer;
 sitios2,edificios2:conjunto;
 Q:sol;
begin
{*** Inicia construcción de una solución ***}
{*** factible X a partir de T.           ***}
 repeat
  repeat
   j:=trunc(random*n)+1;
  until j in sitios;
  case j of
   1..14:begin
```

```
       sitios2:=[1..14]*sitios;
       edificios2:= [1..14]*edificios;
       de:=1;
       a:=14;
      end;
15..18:begin
        sitios2:=[15..18]*sitios;
        edificios2:=[15..18]*edificios;
        de:=15;
        a:=18;
       end;
19..23:begin
        sitios2:=[19..23]*sitios;
        edificios2:=[19..23]*edificios;
        de:=19;
        a:=23;
       end;
24,25:begin
       sitios2:=[24..25]*sitios;
       edificios2:=[24..25]*edificios;
       de:=24;
       a:=25;
      end;
26,27:begin
       sitios2:=[26..27]*sitios;
       edificios2:=[26..27]*edificios;
       de:=26;
       a:=27;
      end;
28,29:begin
       sitios2:=[28..29]*sitios;
       edificios2:=[28..29]*edificios;
       de:=28;
       a:=29;
      end;
30,31:begin
       sitios2:=[30..31]*sitios;
       edificios2:=[30..31]*edificios;
```

```
        de:=30;
        a:=31;
       end;
 32,33:begin
        sitios2:=[32..33]*sitios;
        edificios2:=[32..33]*edificios;
        de:=32;
        a:=33;
       end;
 34,35:begin
        sitios2:=[34..35]*sitios;
        edificios2:=[34..35]*edificios;
        de:=34;
        a:=35;
       end;
 36,37: begin
         sitios2:=[36..37]*sitios;
         edificios2:=[36..37]*edificios;
         de:=36;
         a:=37;
        end;
     38:begin
         X[j,j]:=1;
         exclude(sitios,j)
        end;
  39,40:begin
         sitios2:=[39..40]*sitios;
         edificios2:=[39..40]*edificios;
         de:=39;
         a:=40;
        end;
end;
if j<>38 then
 begin
  m:=a-de+1;
  sitios:=sitios-sitios2;
  construSolFact(sitios2,edificios2,j,de,a,m,X,T)
 end;
```

```
 until sitios=[];
 F:=suma(F,X,n);
 {*** Los tres siguientes renglones transforman la matriz de ***}
 {*** permutación X en la permutación que le correponde, P.  ***}
 for i:=1 to n do
  for j:=1 to n do
   if X[i,j]=1 then P[i]:=j;
 {*** Termina construcción de una solución factible X a ***}
 {*** partir de T.                                      ***}
 {*** Aplicación del operador a la solución obtenida.   ***}
 operadorP(n,0,E,P,Q);
 P:=Q.P;
 v1:=Q.valor;
 cambiarSiMejoraAlPeor(E,n,Q.valor,Q.P,'10 CombSol ',cambio);
end;

procedure combinaSol(var E:elite;var F:matriz;const n:Integer;
              var P:permutacion;var v1:integer;var X:matriz01;
              var sitios,edificios:conjunto;var cambio:boolean);
var
 ordenE:conjunto;
 i,j,k:Integer;
 T:matriz;
 matrizCero:matriz01;
begin
 for i:=1 to n do
  for j:=1 to n do
   begin
    T[i,j]:=0;
    MatrizCero[i,j]:=0;
   end;
 Randomize;
 {*** Inicia obtención de una solución combinada T ***}
 repeat
  j:=trunc(random*R)+1; {R=5 máximo de soluciones a combinar}
 until j>1;
 {* Se eligen j matrices de permutaciones y se suman estas, *}
 {* la suma esta en T                                      *}
```

```
 if n>=20 then
  begin
   ordenE:=[1..20];
   k:=trunc(random*20)+1;
   F:=suma(F,E[k].MP,n);
   T:=suma(T,E[k].MP,n);
   for i:=2 to j do
    begin
     exclude(ordenE,k);
     repeat
      k:=trunc(random*20)+1;
     until k in ordenE;
     F:=suma(F,E[k].MP,n);
     T:=suma(T,E[k].MP,n);
    end;
  end; {if}
 {*** Termina obtención de una solución combinada T ***}
 construPermuFact(T,F,n,sitios,edificios,X,P,v1,cambio,E);
end; {end procedure combinaSol}

procedure intensificacion(var E:elite;var F:matriz;const n:
                integer;var v1:Integer; var P:permutacion;
                const cs:boolean;var cambio:boolean);
type
 lista=array[0..7] of permutacion;
var
 L:lista;
 i,j,k,m,q,indi,tope,v2:integer;
 X:matriz01;
 sitios,edificios:conjunto;
 esta:boolean;
begin
 for i:=1 to n do
  for j:=1 to n do
   X[i,j]:=0;
 sitios:=[1..n];
 edificios:=[1..n];
 if cs then combinaSol(E,F,n,P,v1,X,sitios,edificios,cambio);
```

```
tope:=1200;
L[0]:=P;
for m:=1 to 7 do
 begin
  repeat
   indices(j,k,n);
   L[m]:=L[m-1];
   L[m][j]:=L[m-1][k];
   L[m][k]:=L[m-1][j];
   indi:=0;
   esta:=false;
   while (m>1) and (indi<m) and (not esta) do
    begin
     q:=1;
     esta:=(L[m][q]=L[indi][q]);
     while esta and (q<n) do
      begin
       q:=q+1;
       esta:=(L[m][q]=L[indi][q])
      end;
     if (not esta) then indi:=indi+1;
    end; {while (m>1) and (indi<m) and ...}
  until (not esta);
  v2:=evaluaPer(A,B,L[m],n);
  cambiarSiMejoraAlPeor(E,n,v2,L[m],'20 intensi ',cambio);
 end; {for m:=1 to 7 ...}
m:=7;
esta:=false;
for i:=8 to tope do
 begin
  repeat
   if (not esta) then m:=(m+1) mod 8;
   indices(j,k,n);
   if m=0 then
    begin
     L[0]:=L[7];
     L[0][j]:=L[7][k];
     L[0][k]:=L[7][j]
```

```
      end
     else
      begin
       L[m]:=L[m-1];
       L[m][j]:=L[m-1][k];
       L[m][k]:=L[m-1][j]
      end;
     esta:=false;
     indi:=0;
     repeat
      while (not esta) and (indi<>m) and (indi<8) do
       begin
        q:=1;
        esta:=(L[m][q]=L[indi][q]);
        while esta and (q<n) do
         begin
          q:=q+1;
          esta:=(L[m][q]=L[indi][q])
         end;
        if (not esta) then indi:=indi+1;
       end; {while (not esta) ...}
      indi:=indi+1;
     until (indi>7) or esta;
    until (not esta);
    v2:=evaluaPer(A,B,L[m],n);
    cambiarSiMejoraAlPeor(E,n,v2,L[m],'20 intensi ',cambio);
   end {for i:=8 to ...}
end; {procedure intensificacion...}

procedure diversificacion(var E:elite;var F:matriz;const n:
      integer;var P:permutacion; var X:matriz01;var sitios,
      edificios:conjunto;var v1:integer; var cambio:boolean);
var
 i,j,k,l,m,de,a,m2:integer;
 Z:permuta;
 eleg,sitios2,edificios2:conjunto;
 matrizCero:matriz01;
 Q:sol;
```

```
 T:matriz;
begin
 m:=round(0.05*n);
 sitios:=[1..n];
 edificios:=[1..n];
 sitios2:=[1..14];
 edificios2:=[1..14];
 for i:=1 to n do
  for j:=1 to n do
   begin
    X[i,j]:=0;
    matrizCero[i,j]:=0
   end;
 de:=1;
 a:=14;
 m2:=a-de+1;
 for l:=1 to m do
  begin
   repeat
    j:=trunc(random*n)+1;
   until j in sitios2;
   exclude(sitios2,j);
   for i:=de to a do
    begin
     Z[i].v:=F[j,i];
     Z[i].p:=i
    end;
   ordenaPer(Z,de+1,a);
   k:=1;
   while not(Z[k].p in edificios2) do k:=k+1;
   eleg:=[];
   for i:=de to a do
    if (Z[k].v=Z[i].v) and (Z[i].p in edificios2) then
       include(eleg,Z[i].p);
   repeat
    i:=trunc(random*m2)+de
   until i in eleg;
   X[j,i]:=1;
```

```
    exclude(edificios2,i);
   end; {end for l:=1...}
   sitios:=sitios2+[15..40];
   edificios:=edificios2+[15..40];
   combinaSol(E,F,n,P,v1,X,sitios,edificios,cambio);
   Q.valor:=v1;
   Q.P:=P;
   Q.MP:=matrizCero;
   for j:=1 to n do Q.MP[Q.P[j],j]:=1;
   if n>=20 then m:=20;
   for l:=1 to m do
    begin
     F:=suma(F,Q.MP,n);
     T:=sumaMat(Q.MP,E[l].MP,n);
     sitios:=[1..n];
     edificios:=[1..n];
     X:=matrizCero;
     construPermuFact(T,F,n,sitios,edificios,X,P,v1,cambio,E);
    end;
end;

procedure busquedaD(const n:integer;const archivo:
                    string255;var E:elite);
var
 i,j,k,v1,prom,mejor,corridas,iteSinM,promIte,mayor:Integer;
 F:matriz; {F matriz de frecuencias}
 P:permutacion;
 s:string;
 X,matrizCero:matriz01;
 sitios,edificios:conjunto;
 hora:TDateTime;
 cambio:boolean;
begin
 AssignFile(archi,
           'C:\carmen\tesisMaestria\progDelphi\mejora.txt');
 Rewrite(archi);
 leeMatrices(A,B,n,archivo);
 corridas:=2;
```

```
hora:=Time;
Form1.Caption:=TimeToStr(hora);
writeln(archi,TimeToStr(hora));
prom:=0;
promIte:=0;
for k:=1 to corridas do
 begin
  pobIni(E,n);
  for i:=1 to n do
   for j:=1 to n do
    begin
     F[i,j]:=0;
     MatrizCero[i,j]:=0;
    end;
  cambio:=false;
  iteSinM:=0;
  i:=0;
  while (iteSinM<23) and (i<500) do
   begin
    i:=i+1;
    writeln(archi,'Iteracion ',IntToStr(i));
    if (i mod 9)=8 then
     begin
      case n of
       19..26:intensificacion(E,F,n,v1,P,true,cambio);
       27..49:intensificacion(E,F,n,E[1].valor,E[1].P,false,cambio)
      end;
     end
    else
     if (i mod 9)=0 then
      begin
       writeln(archi,'Pob');
       if n>=20 then
        for j:=1 to 20 do writeln(archi,IntToStr(E[j].valor));
       diversificacion(E,F,n,P,X,sitios,edificios,v1,cambio);
       if cambio then
        begin
         writeln(archi,'Pob');
```

```
        if n>=20 then
         for j:=1 to 20 do
          writeln(archi,IntToStr(E[j].valor));
        end
      end
     else
      begin
       sitios:=[1..n];
       edificios:=[1..n];
       X:=matrizCero;
       combinaSol(E,F,n,P,v1,X,sitios,edificios,cambio);
       if (i mod 18)=1 then mejora(n,0,E,cambio);
      end;
    writeln(archi,'E[1].valor= ',IntToStr(E[1].valor));
    if cambio then
     begin
      cambio:=false;
      iteSinM:=0;
     end
    else
     iteSinM:=iteSinM+1;
   end; {while...}
  hora:=Time;
  s:=";
  for j:=1 to n do s:=s+IntToStr(E[1].P[j])+' ';
  writeln(archi,TimeToStr(hora),'E[1].P=',s);
  writeln(archi,'iteSinM=',IntToStr(iteSinM));
  prom:=prom+E[1].valor;
  promIte:=promIte+i;
  if k=1 then
   begin
    mejor:=E[1].valor;
    mayor:=i;
   end
  else
   begin
    if E[1].valor<mejor then mejor:=E[1].valor;
    if i>mayor then mayor:=i
```

```
    end;
  end; {for k:=1 to ...}
  prom:=round(prom/corridas);
  promIte:=round(promIte/corridas);
  for i:=1 to n do
   begin
    for j:=1 to n-1 do
     write(archi,IntToStr(F[i,j]),' ');
    writeln(archi,IntToStr(F[i,n]))
   end;
  ShowMessage('Promedio='+IntToStr(prom)+' Mejor='+IntToStr(mejor)+
              ' promIte='+IntToStr(promIte)+' maxIte='+
              IntToStr(mayor)+' '+TimeToStr(hora));
  writeln(archi,'Promedio=',IntToStr(prom),' Mejor=',IntToStr(mejor),
          ' promIte=',IntToStr(promIte),' maxIte= ',IntToStr(mayor));
  CloseFile(archi);
end;

procedure evalPermu(const archivo,archivo2:string255);
var
 valor:integer;
 P:permutacion;
begin
 leeMatrices(A,B,40,archivo2);
 leePermu(P,40,archivo);
 valor:=evaluaPer(A,B,P,40);
 ShowMessage('El valor de la permutación es: '+IntToStr(valor));
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
 busquedaD(40,'C:\carmen\tesisMaestria\progDelphi\cia40d.txt',E)
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
 busquedaD(40,'C:\carmen\tesisMaestria\progDelphi\cia40c.txt',E)
end;
```

```
procedure TForm1.Button3Click(Sender: TObject);
begin
 evalPermu('C:\carmen\tesisMaestria\progDelphi\original.txt',
           'C:\carmen\tesisMaestria\progDelphi\cia40d.txt');
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
 evalPermu('C:\carmen\tesisMaestria\progDelphi\optimaMod.txt',
           'C:\carmen\tesisMaestria\progDelphi\cia40d.txt');
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
 evalPermu('C:\carmen\tesisMaestria\progDelphi\optima.txt',
           'C:\carmen\tesisMaestria\progDelphi\cia40d.txt');
end;

procedure TForm1.Button6Click(Sender: TObject);
begin
 evalPermu('C:\carmen\tesisMaestria\progDelphi\optimaLimac.txt',
           'C:\carmen\tesisMaestria\progDelphi\cia40d.txt');
end;

procedure TForm1.Button7Click(Sender: TObject);
begin
 evalPermu('C:\carmen\tesisMaestria\progDelphi\optimoCia40b.txt',
           'C:\carmen\tesisMaestria\progDelphi\cia40b.txt');
end;
end.
```

Bibliografía

[ADA2001] Adams, W. P., Guignard, M., Hahn, P. M., Hightower, W. L., *A level-2 reformulation-linearization technique bound for the quadratic assignment problem*, European Journal of Operational Research 180, pp. 983-996, 2007. Primero disponible como Working Paper 01-04, Systems Engineering Department, University of Pennsylvania (Authors: P.M. Hahn, W.L. Hightower, T.A. Johnson, M. Guignard, and C. Roucairol), 2001.

[AHO74] Aho, A. V.,Hopcroft, J. E., and Ullman J. D., *The Design and Analysis of Computers Algorithms,* Addison-Wesley, 1974.

[AHU2000] Ahuja, R. K. Orlin, J. B., Tiwari, A., *A greedy genetic algorithm for the quadratic assignment problem*, Computers and Operations Research 27 (10), pp. 917-934, 2000.

[AND96] Anderson, E. J., *Theory and methodology: Mechanisms for local search*, European Journal of Operational Research 88 (1), pp.139-151, 1996.

[ANG98] Angel, E., Zissimopoulos, V., *On the quality of local search for the quadratic assignment problem*, Discrete Applied Mathematics 82 (1-3), PP. 15-25, 1998.

[ANG2000] Angel, E., Zissimopoulos, V., *On the classification of NP-complete problems in terms of their correlation coefficient*, Discrete Applied Mathematics 99 (1-3), pp. 261-277, 2000.

[ANG2001] Angel, E., Zissimopoulos, V., *On the landscape ruggedness of the quadratic assignment problem*, Theoretical Computer Science 263 (1-2), pp. 159-172, 2001.

[ANG2002] Angel, E., Zissimopoulos, V., *On the hardness of the quadratic assignment problem with metaheuristics*, Journal of Heuristics 8 (4), pp. 399–414, 2002.

[ANS2001] Anstreicher, K.M., Brixius, N. W., *A new bound for the quadratic assignment problem based on convex quadratic programming*, Mathematical Programming 89 (3), pp. 341-357, 2001.

[ANS2002] Anstreicher, K. M., Brixius, N. W., Goux, J. P., Linderoth, J., *Solving large quadratic assignment problems on computational grids*, Mathematical Programming 91, (3), pp. 563-588, 2002.

[ARM63] Armour, G. C., Buffa, E. S., A heuristic algorithm and simulation approach to relative location of facilities, Management Science 9 (2), pp. 294-309, 1963.

[BAL2003] Balakrishnan, J., Cheng, C. H., Conway, D. G., Lau, C. M., A hybrid genetic algorithm for the dynamic plant layout problem, International Journal of Production Economics 86 (2), pp. 107-120, 2003.

[BAR2003] Barvinok, A., Stephen, T., *The distribution of values in the quadratic assignment problem*, Mathematics of Operations Research 28, pp. 64-91, 2003.

[BAT94] Battiti, R., Tecchiolli, G., *Simulated annealing and tabu search in the long run: A comparison on QAP tasks*, Computers and Mathematics with Applications 28 (6), pp. 1-8, 1994.

[BAZ80] Bazaraa, M. S., Sherali, H. D., *Benders' partitioning scheme applied to a new formulation of the quadratic assignment problem*, Naval Research Logistics Quarterly 27, pp. 29-41, 1980.

[BAZ82] Bazaraa, M. S., Sherali, H. D., *On the use of exact and heuristic cutting plane methods for the quadratic assignment problem*, Journal of the Operational Research Society 33 (11), pp. 991-1003,1982.

[BLA91] Bland, J. A., Dawson, G. P., *Tabu search and design optimization*, Computer-Aided Design 23 (3), pp. 195-201,1991.

[BOL96] Bölte, A., Thonemann, U. W., *Optimizing simulated annealing schedules with genetic programming*, European Journal of Operational Research 92 (2), pp. 402-416, 1996.

[BOZ96] Bozer Y. A., Suk-Chul, R., *A branch and bound method for solving the bidirectional circular layout problem*, Applied Mathematical Modelling 20 (5), pp. 342-351, 1996.

[BRI2001] Brixius, N.W., Anstreicher, K.M., *Solving quadratic assignment problems using convex quadratic programming relaxations,* Optimization Methods and Software 16 (1-4), pp. 49-68, 2001.

[BRU84] Bruijs, P. A., *On the quality of heuristic solutions to a 19X19 quadratic assignment problem*, European Journal of Operational Research 17 (1), pp. 21-30, 1984.

[BRU98] Brüngger, A., Marzetta, A., Clausen, J., Perregaard, M., *Solving large-scale QAP problems in parallel with the search library ZRAM*, Journal of Parallel and Distributed Computing 50, pp.157-169, 1998.

[BUR80] Burkard, R. E., Derigs, U., *Assignment and matching problems: Solution methods with Fortran programs*, Lecture notes in economics and mathematical systems, vol. 184, Springer-Verlag, Berlin-New York,1980.

[BUR84] Burkard, R. E., Rendl, F., *A thermodynamically motivated simulation procedure for combinatorial optimization problems*, European Journal of Operational Research 17 (2), pp. 169-174, 1984.

[BUR91] Burkard, R. E., *Locations with spatial interactions: The quadratic assignment problem* en Mirchandani, P. B., Francis, R. L. Editors, Discrete Location Theory, John Wiley and Sons, pp. 387-437, 1991.

[BUR97] Burkard, R. E., Karisch, S. E., and Rendl, F., *QAPLIB-A Quadratic Assignment Problem Library*, Journal of Global Optimization 10 (4), pp. 391-403, 1997.

[BV2004] Burer, S., Vandenbussche, D., *Solving lift-and-project relaxations of binary integer programs*, SIAM Journal on Optimization 16, pp. 726-750, 2006. Primero disponible en *http://www.optimization-online.org/DB_HTML/2004/06/890.html*, 2004.

[CAR92] Carraresi, P., Malucelli, F., *A new lower bound for the quadratic assignment problem*, Operations Research 40, Supplement 1, pp. S22-S27, 1992.

[CAR93] Carraresi, P., Malucelli, F., *A reformulation scheme and new lower bounds for the QAP* en DIMACS Series in Discrete Mathematics and Theoretical Computers Science, vol. 16, Quadratic Assignment and Related Problems, Pardalos, P. M., Wolkowicz, H. Editors, Rhode Island, pp. 147-160, 1993.

[CEL98] Çela, E., *The quadratic assignment problem: Theory and algorithms* en Combinatorial Optimization, Kluwer Academic Publishers, Dordrecht, 1998.

[CHA93] Chakrapani, J., Skorin-Kapov, J., *Massively parallel tabu search for the quadratic assignment problem*, Annals of Operations Research 41 (4), pp. 327-341, 1993.

[CHE95] Chen, B.,*Special cases of the quadratic assignment problem*, European Journal of Operational Research 81 (2), pp. 410-419, 1995.

[CHI98] Chiang, W. C., Chiang, C., *Intelligent local search strategies for solving facility layout problems with the quadratic assignment problem formulation*, European Journal of Operational Research 106 (2-3), pp. 457-488, 1998.

[CHR81] Christofides, N., Gerrard, M., *A graph theoretic analysis of bounds for the quadratic assignment problem* en Studies on graphs and discrete programming, Ed. Hansen, P., North-Holland, pp. 61-68, 1981.

[CHR89] Christofides, N., Benavent, E., *An exact algorithm for the quadratic assignment problem on a tree*, Operations Research 37 (5), pp. 760-768, 1989.

[CLA96] Clausen, J., Perregaard, M., *On the best search strategy in parallel branch-and-bound - Best-first-search vs. lazy dept-first-search*, Annals of Operations Research 11, pp. 1-17, 1996.

[CLA97] Clausen, J., Perregaard, M., *Solving large quadratic assignment problems in parallel*, Computational Optimization and Applications 8 (2), pp. 111-127, 1997.

[CON90] Connolly, D. T., *An improved annealing scheme for the QAP*, European Journal of Operational Research 46 (1), pp. 93-100, 1990.

[COR2001] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C., *Introduction to Algorithms,* MIT Cambridge, Massachusetts, 2nd ed., 2001.

[CUN97] Cung, Van-Dat, Mautor, Thierry, Michelon, Philippe, Tavares, Andréa, *A Scatter Search Based Approach for the Quadratic Assignment Problem*, Proceeding of the IEEE Conference on Evolutionary Computation, ICEC, pp. 165-169, 1997.

[CYG94] Cyganski, D., Vaz, R. F., Virball, V. G., *Quadratic assignment problems generated with the Palubetskis algorithm are degenerate*, IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications 41 (7), pp. 481-484, 1994.

[DAV87] Davis, L., *Genetic algorithms and simulated annealing*, Morgan Kaufman Publishers, Los Altos, California, USA, 1987.

[DEI2000] Deineko, V. G., Woeginger, G. J., *A study of exponential neighborhoods for the traveling salesman problem and for the quadratic assignment problem*, Mathematical Programming 87 (3), pp. 519-542, 2000.

[DIC72] Dickey, J. W., Hopkins, J. W., *Campus building arrangement using Topaz*, Transportation Research 6, pp. 59-68, 1972.

[DOR96] Dorigo, M., Maniezzo, V., Colorni, A., *The ant system: Optimization by a colony of cooperating agents*, IEEE Transactions on systems, man, and cybernetics-Part B 26 (1), pp. 29-41, 1996.

[DRE2003] Drezner, Z., *A new genetic algorithm for the quadratic assignment problem*, Informs Journal on Computing 15 (3), pp. 320-330, 2003.

[DRE2005] Drezner, Z., *Compounded genetic algorithms for the quadratic assignment problem*, Operations Research Letters 33 (5), pp. 475-480, 2005.

[DUN2004] Dunker, T., Radons, G., Westkämper, E., *Combining evolutionary computation and dynamic programming for solving a dynamic facility layout problem*, European Journal of Operational Research 165 (1), pp. 55-69, 2005. Disponible en línea desde el 12 de marzo de 2004.

[ELB2004] El-Baz, M. A., *A genetic algorithm for facility layout problems of different manufacturing environments*, Computers and Industrial Engineering 47 (2-3), pp. 233 – 246, 2004.

[ELS77] Elshafei, A. N., *Hospital layout as a quadratic assignment problem*, Operations Research Quarterly 28 (1), pp. 167-179, 1977.

[FEO95] Feo,T. A., Resende, M. G. C., *Greedy randomized adaptive search procedures*, Journal of Global Optimization 6 (2), pp. 109-133, 1995.

[FLE94] Fleurent, C., Ferland, J. A., *Genetic hybrids for the quadratic assignment problem* en DIMACS Series in Discrete Mathematics and Theoretical Computers Science, vol. 16, Quadratic Assignment and Related Problems, Pardalos, P. M., Wolkowicz, H. Editors, American Mathematical Society, Providence, Rhode Island, pp. 173 – 187, 1994.

[FLE99] Fleurent, C., Glover, F., *Improved constructive multistart strategies for the quadratic assignment problem using adaptative memory*, INFORMS Journal on Computing 11 (2), pp. 198–204, 1999.

[FIS88] Fisher, M. L., and Rinnooy Kan, *The design, analysis and implementation of heuristics*, Management Science 34, pp. 263-265, 1988.

[FOR94] Forsberg, J. H., Delaney, R. M., Zhao, Q., Harakas, G.,Chandran, R., *Analyzing lanthanide-induced shifts in the NMR spectra of lanthanide(III) complexes derived from 1,4,7,10-tetrakis(N,N-diethylacetamido)-1,4,7,10-tetraazacyclododecane*, Inorganic Chemistry 34, pp.3705-3715, 1994.

[FRI83] Frieze, A. M., Yadegar, J. , *On the quadratic assignment problem*, Discrete Applied Mathematics 5, pp. 89-98, 1983.

[GAL2006] Gallego Rendón, R. A., Escobar Zuluaga, A., y Romero Lazaro, R. A., *Técnicas de optimización combinatorial*, Universidad Tecnológica de Pereira, Colombia, 2006.

[GAM99] Gambardella, L. M., Taillard, É. D., Dorigo, M., *Ant colonies for the quadratic assignment problem*, Journal of the Operational Research Society 50, pp. 167 – 176, 1999.

[GAR79] Garey, M. R., and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[GIL62] Gilmore, P. C., *Optimal and suboptimal algorithms for the quadratic assignment problem*, Society for Industrial and Applied Mathematics, Journal of the Society for Industrial and Applied Mathematics 10, pp. 305-313, 1962 .

[GLO77] Glover, F., *Heuristics for integer programming using surrogate constraints*, Decision Science 8 (1), pp. 156–166, 1977.

[GLO89] Glover, F., *Tabu search—Part I*, ORSA Journal on Computing 1 (3), pp. 190-206, 1989.

[GLO90] Glover, F., *Tabu search—Part II*, ORSA Journal on Computing 2 (1), pp. 4-32, 1990.

[GOL89] Goldberg, D. E., *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley Longman Publishing, Boston, Massachusetts, USA, 1989.

[GON2000] González Velásquez, Rogelio, *GRASP en Paralelo para el Problema de Asignación Cuadrática*, Tesis de maestría, DEPFI-UNAM, México, 2000.

[GRA70] Graves, G. W.,Whinston, A. B., *An algorithm for the quadratic assignment problem*, Management Science 17 (7), pp. 453-471, 1970.

[GUT91] Gutiérrez Andrade, M. A., *La técnica de recocido simulado y sus aplicaciones,* Tesis de doctorado, DEPFI-UNAM, México, 1991.

[GUT2002] Gutin, G., Yeo, A., *Polynomial approximation algorithms for TSP and QAP with a factorial domination numbers*, Discrete Applied Mathematics 119 (1-2), pp. 107-116, 2002.

[HAH98] Hahn, P., Grant, T., *Lower bounds for the quadratic assignment problem based upon a dual formulation*, Operations Research 46, pp.912-922, 1998.

[HAH2000] Hahn, P., *Progress in solving the Nugent instances of the quadratic assignment problem*, Working Paper, Systems Engineering, University of Pennsylvania, 2000.

[HAH2001] Hahn, P., Hightower, W. L., Johnson, T. A., Guignard-Spielberg, M., Roucairol, C.,*Tree elaboration strategies in branch-and-bound algorithms for solving the quadratic assignment problem*, Yugoslav Journal of Operations Research 11 (1), pp. 41-60, 2001.

[HAH2004] Hahn, P., Kim, B.-J., Stützle, T., Kanthak, S., Hightower, W. L., Samra, H., Ding, Z., Guignard, M., *The quadratic three-dimensional assignment problem: Exact and approximate solution methods*, European Journal of Operational Research 184 (2), 16, pp. 416-428, 2008. Primero disponible como *The quadratic three-dimensional assignment problem: Exact and heuristic solution methods*, Working Report No. 04-08-02, The Wharton School, University of Pennsylvania, Philadelphia, Pennsylvania, USA, 2004.

[HAS2002] Hasegawa, M., Ikeguchi, T., Aihara, K., Itoh, K., *A novel chaotic search for quadratic assignment problems*, European Journal of Operational Research 139 (3), pp. 543-556, 2002.

[HEI73] Heider, C. H., *An N-step, 2-variable search algorithm for the component placement problem*, Naval Research Logistics Quarterly 20 (4), pp. 699–724, 1973.

[HER85] Herroelen, W., Van Gils, A., *On the use of flow dominance in complexity measures for facility layout problems*, International Journal of Production Research 23 (1), pp. 97-108, 1985.

[HUN91] Huntley, C. L., Brown, D. E., *A parallel heuristic for quadratic assignment problems*, Computers and Operations Research 18 (3), pp. 275-289, 1991.

[HUN96] Huntley, C. L., Brown, D. E., *Parallel genetic algorithms with local search*, Computers and Operations Research 23 (6), pp. 559-571, 1996.

[ISH98] Ishii, S., Sato, M., *Constrained neural approaches to quadratic assignment problems*, Neural Networks 11 (6), pp. 1073-1082, 1998.

[ISH2001] Ishii, S., Sato, M., *Doubly constrained network for combinatorial optimization*, Neurocomputing 43 (1-4), pp. 239-257, 2002. Disponible en línea desde el 14 de diciembre de 2001.

[JOH97] Johnsonbaugh, Richard, *Discrete Mathematics*, Prentice Hall, New Jersey, 4th ed., 1997.

[JUN2000] Jünger, M., Kaibel, V., *On the SQAP-polytope*, SIAM Journal on Optimization 11(2), pp. 444-463, 2000.

[JUN2001a] Jünger, M., Kaibel, V., *The QAP-polytope and the star transformation*, Discrete Applied Mathematics 111 (3), pp. 283-306, 2001a.

[JUN2001b] Jünger, M., Kaibel, V., *Box-inequalities for quadratic assignment polytopes*, Mathematical Programming 91, pp. 175-197, 2001b.

[KAI98] Kaibel, V., *Polyhedral combinatorics of quadratic assignment problems with less objects than locations*, Lecture Notes in Computer Science 1412, pp. 409-422, 1998.

[KAU78] Kaufman, L., Broeckx, F., *An algorithm for the quadratic assignment problem using Bender's decomposition*, European Journal of Operational Research 2 (3), pp. 207-211, 1978.

[KIR83] Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P., *Optimization by simulated anneling*, Science 220, pp. 671-680, 1983.

[KOC98] Kochhar, J. S., Foster, B. T., Heragu, S. S., *Hope: A genetic algorithm for the unequal area facility layout problem*, Computers and Operations Research 25 (7-8), pp. 583-594, 1998.

[KOO57] Koopmans, T. C. and Beckmann, M. J., *Assignment Problems and the Location of Economic Activities*, Econometrica 25, pp. 53-76, 1957.

[KOR2000] Korte, B., and Vygen J., *Combinatorial Optimization: Theory and Algorithms,* Springer-Verlag Berlin Heidelberg New York, 2000.

[LAW63] Lawler, E. L., *The quadratic assignment problem*, Management Science 9, pp. 586-599, 1963.

[LI92] Li, Y., Pardalos, P.M., *Generating quadratic assignment test problems with known optimal permutations*, Computational Optimization and Applications 1, pp.163-184, 1992.

[LI94] Li, Y., Pardalos, P. M., Resende, M. G. C., *A greedy randomized adaptive search procedure for the quadratic assignment problem* en DIMACS Series in Discrete Mathematics and Theoretical Computers Science, vol. 16, Quadratic Assignment and Related Problems, Pardalos, P. M., Wolkowicz, H. Editors, American Mathematical Society, Providence, Rhode Island, pp. 237-261, 1994.

[LIA96] Liang, Y., *Combinatorial optimization by Hopfield networks using adjusting neurons*, Information Sciences 94 (1-4), pp. 261-276, 1996.

[LIM2000] Lim, M. H., Yuan, Y., Omatu, S., *Efficient genetic algorithms using simple genes exchange local search policy for the quadratic*

assignment problem, Computational Optimization and Applications 15 (3), pp. 249 – 268, 2000.

[LIM2002] Lim, M. H., Yuan, Y., Omatu, S., *Extensive testing of a hybrid genetic algorithm for solving quadratic assignment problems*, Computational Optimization and Applications 23 (1), pp. 47 – 64, 2002.

[LOI2007] Loiola, E. M., de Abreu, N. M. M., Boaventura-Netto, P. O., Hahn, P., Querido, T., *A survey for the quadratic assignment problem,* European Journal of Operational Research 176 (2), pp. 657-690, 2007.

[MAN95] Maniezzo, V., Dorigo, M., Colorni, A., Algodesk: *An experimental comparison of eight evolutionary heuristics applied to the quadratic assignment problem*, European Journal of Operational Research 81 (1), pp. 188-204, 1995.

[MAN99] Maniezzo, V., Colorni, A., *The ant system applied to the quadratic assignment problem*, IEEE Transactions on Knowledge and Data Engineering 11 (5), pp. 769 – 778, 1999.

[MAU94] Mautor, T., Roucairol, C., *Difficulties of exact methods for solving the QAP* en DIMACS Series in Discrete Mathematics and Theoretical Computers Science, vol. 16, Quadratic Assignment and Related Problems, Pardalos, P. M., Wolkowicz, H. Editors, American Mathematical Society, Providence, Rhode Island, pp. 263-274, 1994.

[MAV97] Mavridou, T. D., Pardalos, P. M., *Simulated annealing and genetic algorithms for the facility layout problem: A survey*, Computational Optimization and Applications 7 (1), pp. 111-126, 1997.

[MER97] Merz, P., Freisleben, B., *A genetic local search approach to the quadratic assignment problem* en Proceedings of the 7th International Conference on Genetic Algorithms, Morgan Kaufmann, pp. 465 – 472, 1997.

[MER99] Merz, P., Freisleben, B., *A comparison of memetic algorithms, tabu search, and ant colonies for the quadratic assignment problem* en Proceedings of the 1999 Congress on Evolutionary Computation (CEC 99), IEEE, pp. 2063 – 2070, 1999.

[MER2000] Merz, P., Freisleben, B., *Fitness landscape analysis and memetic algorithms for the quadratic assignment problem*, IEEE Transactions on Evolutionary Computation 4 (4), pp. 337 – 352, 2000.

[MID2002] Middendorf, M., Reischle, F., Schmeck, H., *Multi colony ant algorithms*, Journal of Heuristics 8 (3), pp. 305-320, 2002.

[MIR2005] Miranda, G., Luna, H. P. L., Mateus, G. R., Ferreira, R. P. M., *A performance guarantee heuristic for electronic components placement problems including thermal effects.* Computers and Operations Research 32 (11), pp. 2937-2957, 2005.

[MIS97] Misevicius, A., *A new algorithm for the quadratic assignment problem*, Information technology and control, Kaunas, Technologija 5, pp. 39- 44, 1997.

[MIS99] Misevicius, A., Riskus, A., *Multistart threshold accepting: experiments with the quadratic assignment problem*, Information technology and control, Kaunas, Technologija 12, pp. 31 – 39, 1999.

[MIS2003] Misevicius, A., *A modified simulated annealing algorithm for the quadratic assignment problem*, Informatica 14 (4), pp. 497-514, 2003.

[MIS2004] Misevicius, A., *An improved hybrid genetic algorithm: new results for the quadratic assignment problem*, Knowledge-Based Systems 17 (2-4), pp. 65-73, 2004.

[MLA97] Mladenovic, N., Hansen, P., *Variable neighborhood search*, Computers and Operations Research 24 (11), pp. 1097 – 1100, 1997.

[MOE2003] Moe, R., *GRIBB-Branch-and-bound methods on the Internet*, en Parallel Processing and Applied Mathematics 2003, Lecture Notes in Computer Science 3019, pp. 1020-1027, 2003.

[NIS94] Nissen, V., *Solving the quadratic assignment problem with clues from nature*, IEEE Transactions on Neural Networks 5 (1), pp. 66-72, 1994.

[NIS97] Nissen, V., *Quadratic assignment* en Handbook of Evolutionary Computation, Back, T., Fogel, D. B., Michalewicz, Z. Editors, pp. 1 – 8, 1997.

[NUR68] Nugent C. E., Vollmann,T. E. and Ruml, J., *An Experimental Comparison of Techniques for Assignment of Facilities to Locations*, Operations Research 16, pp. 150-173, 1968.

[PAD91] Padberg, M., Rinaldi, G., *A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems*, SIAM Review 33 (1), pp. 60-100, 1991.

[PAD96] Padberg, W., Rijal, P., *Location, scheduling, design, and integer programming*, Kluwer Academic Publishers, Boston, 1996.

[PAR97] Pardalos, P. M., Ramakrishnan, K. G., Resende, M. G. C., Li, Y., *Implementation of a variance reduction-based lower bound in a branch-and-bound algorithm for the quadratic assignment problem*, SIAM Journal on Optimization 7 (1), pp. 280-294, 1997.

[POL76] Pollatschek, M. A. Gershoni, N., Radday, Y. T.,*Optimization of the typewriter keyboard by simulation*, Angewandte Informatik. v17, pp. 438-439, 1976.

[RAB2003] Rabak, C. S., Sichman, J. S., *Using A.Teams to optimize automatic insertion of electronic components.* Advanced Engineering Informatics 17 (2), pp. 95-106, 2003.

[RAM2002] Ramakrishnan, K.G., Resende, M. G. C., Ramachandran, B., Pekny, J. F., *Tight QAP bounds via linear programming* en Pardalos, P.M., Migdalas, A. Burkard R. E. Editors, Combinatorial and Global Optimization, World Scientific Publishing, Singapore, pp. 297-303, 2002.

[REE93] Reeves, C. R., *Modern heuristic techniques for combinatorial problems*, New York, New York: Halsted, 1993.

[REN2003] Rendl, F., Sotirov, R., *Bounds for the quadratic assignment problem using the bundle method*, Mathematical Programming 109, pp. 505-524, 2007. Primero disponible como Technical Report, Department of Mathematics, University of Klagenfurt, 2003.

[RES95] Resende, M. G. C., Ramakrishnan, K. G., Drezner, Z., *Computing lower bounds for the quadratic assignment problem with an interior point algorithm for linear programming*, Operations Research 43, pp. 781-791, 1995.

[ROU79] Roucairol, C., *A reduction method for quadratic assignment problems*, Operations Research Verfahren 32, pp. 183-187, 1979.

[ROU87] Roucairol, C., *A parallel branch and bound algorithm for the quadratic assignment problem*, Discrete Applied Mathematics, 18 (2), pp. 211-225, 1987.

[SAH76] Sahni, S., and González, T., *P-Complete Approximation Problems*, Journal of the Association for Computing Machinery 23 (3), pp. 555-565, 1976.

[SAR95] Sarker, B. R., Wilhelm, W. E., Hogg, G. L., Han, M.-H., *Backtracking of jobs in one-dimensional machine location problems*, European Journal of Operational Research 85 (3), pp. 593-609, 1995.

[SKO90] Skorin-Kapov, J., *Tabu search applied to the quadratic assignment problem*, ORSA Journal on Computing 2 (1), pp. 33-45, 1990.

[SKO94] Skorin-Kapov, J., *Extensions of a tabu search adaptation to the quadratic assignment problem*, Journal Computers and Operations Research 21 (8), pp. 855-865, 1994.

[SPI98] Spiliopoulos, K., Sofianopoulou, S., *An optimal tree search method for the manufacturing systems cell formation problem*, European Journal of Operational Research 105 (3), pp. 537-551, 1998.

[STE61] Steinberg, L., *The backboard wiring problem: A placement algorithm,* Society for Industrial and Applied Mathematics, Review 3, pp. 37-50, 1961.

[STU99] Stützle, T., Dorigo, M., *ACO algorithms for the quadratic assignment problem*, New ideas in optimization, McGraw-Hill Ltd., pp. 33 – 50, 1999.

[STU2000] Stützle, T., Hoos, H., *MAX-MIN Ant system*, Future Generation Computer Systems 16 (9), pp. 889 – 914, 2000.

[SYL87] Sylla, C., Babu, A. J. G., *Methodology for an orderly quadratic assignment problem*, Computers and Industrial Engineering 13 (1-4), pp. 281-284, 1987.

[TAI91] Taillard, E. D., *Robust taboo search for the quadratic assignment problem*, Parallel Computing 17 (4-5), pp. 443-455, 1991.

[TAI95] Taillard, E. D., *Comparison of iterative searches for the quadratic assignment problem*, Location Science 3 (2), pp. 87-105, 1995.

[TAI97] Taillard, E. D., Gambardella, L. M., *Adaptive memories for the quadratic assignment problem*, Technical Report IDSIA-87-97, Lugano, Switzerland, pp. 1-18, 1997.

[TAI2001] Taillard, E., Gambardella, L. M., Gendreau, M., Potvin, J. Y., *Adaptive memory programming: A unified view of metaheuristics*, European Journal of Operational Research 135 (1), pp. 1-16, 2001.

[TAL98] Talbi, E.-G., Hafidi, Z., Geib, J.-M., *A parallel adaptive tabu search approach*, Parallel Computing 24 (14), pp. 2003 – 2019, 1998.

[TAL2001] Talbi, E.-G., Roux, O., Fonlupt, C., Robillard, D., *Parallel ant colonies for the quadratic assignment problem*, Future Generation Computer Systems 17 (4), pp. 441 – 449, 2001.

[TAT95] Tate, D. E., Smith, A. E., *A genetic approach to the quadratic assignment problem*, Computers and Operations Research 22 (1), pp. 73-83, 1995.

[TIA99] Tian, P., Ma, J., Zhang, D.-M., *Application of the simulated annealing algorithm to the combinatorial optimization problem with permutation property: An investigation of generation mechanism*, European Journal of Operational Research 118 (1), pp. 81-94, 1999.

[TSU96] Tsuchiya, K., Bharitkarc, S., Takefuji, *A neural network approach to facility layout problems*, European Journal of Operational Research 89 (3), pp. 556-563, 1996.

[URB98] Urban, T. L., *Solution procedures for the dynamic facility layout problem*, Annals of Operations Research 76, pp. 323-342, 1998.

[UWA2004] Uwate, Y., Nishio, Y., Ushida, A., *Markov chain modeling of intermittency chaos and its application to Hoppfield NN*, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E87-A (4), pp.774-779, 2004.

[WHI94] White, D. J., *Strengthening Gilmore's bound for the quadratic assignment problem*, European Journal of Operational Research 77 (1), pp. 126-140, 1994.

[WIL87] Wilhelm, M. R., Ward, T. L., *Solving quadratic assignment problems by 'simulated annealing'*, IIE Transactions19 (1), pp. 107-119, 1987.

[YIP94] Yip, P.P.C., Pao, Y.-H., *A guided evolutionary simulated annealing approach to the quadratic assignment problem*, IEEE Transactions on Systems, Man and Cybernetics 24 (9), pp. 1383 – 1386, 1994.

[ZAN89] Zanakis, S. H., Evans, J. R. and Vazacopoulos A. A., *Heuristic methods and applications: A categorized survey,* European Journal of Operational Research 43, pp. 88-110, 1989.

[ZHU2007] Zhu, Y-R., *Recent advances and challenges in quadratic assignment and related problems*, disertación para obtener el grado de doctor en filosofía, University of Pennsylvania, 2007