Instituto de Computación - Facultad de Ingeniería
Universidad de la República

Tesis de Maestría en Ingeniería en Computación

Metaheurísticas aplicadas a un Problema de Asignación de Salones y Horarios a Asignaturas

Santiago Costabel

Directora: Maria E. Urquhart

Montevideo, Uruguay
Mayo de 2005

.

Metaheurísticas aplicadas a un Problema de
Asignación de Salones y Horarios a Asignaturas
Santiago Costabel

ISSN 1510-7264
Tesis de Maestría en Ingeniería en Computación
Instituto de Computación - Facultad de Ingeniería
Universidad de la República

Montevideo, Uruguay, Mayo de 2005

A mi vida, Mariana
y a mis pequeños, Jose y Juani,
que me apoyaron en este desafío personal.

Resumen

En la Facultad de Ingeniería, semestre a semestre, la Bedelía se encuentra con el problema de asignarle a cada curso de las diferentes carreras un salón y un horario específico. Este problema es un problema de programación de horarios de difícil solución dada la escasez de recursos que tiene la Facultad, principalmente la disponibilidad de salones.

Los problemas de asignación (de recursos escasos) son problemas de optimización difíciles de resolver en forma exacta. La alternativa es encontrar una solución aproximada para lo que existen métodos heurísticos y metaheurísticas.

Originalmente el problema se resolvía en forma manual por parte de la Bedelía, trabajo que insumía varias semanas. Posteriormente se desarrolló un sistema de información de apoyo a la decisión, y se implementó como método de solución aproximado la metaheurística Tabú Search. A partir de datos existentes, y una solución inicial, interactivamente el usuario obtiene una solución aproximada, que puede ajustar según su propio sentido común y conocimiento de la realidad en la que actúa. Otro trabajo anterior realizó un estudio sobre metaheurísticas para el problema de asignación e implementó Ant System pero de forma independiente y sin los datos de la aplicación real.

En este trabajo se realiza una actualización del estado del arte de los problemas de programación de horarios y se construye un motor de metaheurísticas para la herramienta (software) de apoyo para la asignación de grupos y asignaturas a los salones, que semestralmente realiza la Bedelía de la Facultad de Ingeniería. El motor construido incluye algunas de las metaheurísticas mas importantes como Algoritmos Genéticos, Búsqueda Tabú y dentro de Optimización por Colonia de Hormigas los algoritmos Ant System, Ant Colony System y $\mathcal{MAX} - \mathcal{MIN}$ Ant System.

Finalmente, se presenta la calibración de las diferentes metaheurísticas y la discusión de los resultados obtenidos para instancias reales del problema.

Palabras Claves: Timetabling Problem, Problema de Programación de Horarios, Metaheurísticas.

Índice general

1. Introducción **1**

1.1. Contexto 1

1.2. Motivación 2

1.3. Objetivos 3

1.4. Limitaciones de las Soluciones existentes 3

1.5. Contribución Principal 3

1.6. Organización 4

2. Estado del Arte **5**

2.1. Introducción 5

2.2. Formulación del UCTTP 12

2.3. Métodos y Técnicas de Solución 18

3. Problema de Asignación de Salones y Horarios **35**

3.1. Descripción del Problema 35

3.2. Formalización del PASHA 39

4. Adaptación e Implementación de las Metaheurísticas **47**

4.1. Introducción 47

4.2. Entorno de Trabajo 48

4.3. Problemas Reales 49
4.4. Codificación de la Solución 50
4.5. Restricciones y Preferencias 52
4.6. Función Objetivo 54
4.7. Algoritmos Genéticos 54
4.8. Búsqueda Tabú 59
4.9. Optimización por Colonia de Hormigas 61
4.10. Soluciones Iniciales 68
4.11. Calibración de las Metaheurísticas 69
4.12. Determinación de Valores de los Parámetros 69
4.13. Conclusiones sobre la Calibración de las Metaheurísticas 71

5. Resultados Obtenidos **73**

5.1. Introducción 73
5.2. Datos Reales del PASHA 73
5.3. Plan de Experimentación 74
5.4. Valores de los Parámetros 75
5.5. Problemas 20032-5 y 20041-5 76
5.6. Problemas 20032-5s/P2 y 20041-5s/P2 81
5.7. Análisis de los Resultados 86

6. Conclusiones **91**

6.1. Objetivos Alcanzados 91
6.2. Conclusiones 92
6.3. Evolución del Modelo 93
6.4. Investigaciones y Trabajo Futuro 94

A. Metaheurísticas **97**

A.1. Introducción 97

A.2. Algoritmos Genéticos 98

A.3. Búsqueda Tabú 106

A.4. Optimización por Colonia de Hormigas 108

B. Calibración de las Metaheurísticas **127**

B.1. Método de Calibración 127

B.2. Soluciones iniciales 128

B.3. Calibración de Metaheurísticas 128

Bibliografía **159**

Índice de figuras

4.1. Cromosoma para el PASHA. 55

5.1. Comparación de los Costos obtenidos para 20032-5. 77

5.2. Comparación relativa de los Costos obtenidos para 20032-5. 78

5.3. Comparación de los Costos obtenidos para 20041-5. 79

5.4. Comparación relativa de los Costos obtenidos para 20041-5. 79

5.5. Composición del Costo obtenido para 20032-5. 80

5.6. Composición del Costo obtenido para 20041-5. 80

5.7. Composición en porcentaje del Costo obtenido para 20032-5. 81

5.8. Composición en porcentaje del Costo obtenido para 20041-5. 81

5.9. Comparación de los Costos obtenidos para 20032-5s/P2. 82

5.10. Comparación relativa de los Costos obtenidos para 20032-5s/P2. . . . 83

5.11. Comparación de los Costos obtenidos para 20041-5s/P2. 84

5.12. Comparación relativa de los Costos obtenidos para 20041-5s/P2. . . . 84

5.13. Composición del Costo obtenido para 20032-5s/P2. 85

5.14. Composición del Costo obtenido para 20041-5s/P2. 85

5.15. Composición en porcentaje del Costo obtenido para 20032-5s/P2. . . 86

5.16. Composición en porcentaje del Costo obtenido para 20041-5s/P2. . . 86

A.1. Codificación de los Cromosomas. 101

A.2. Cruzamiento de Cromosomas. 105

A.3. Las hormigas abandonan el hormiguero. 109

A.4. Rastros de feromona de la fuente y del hormiguero. 109

A.5. Ruta más corta. 109

A.6. Ruta alrededor del obstáculo. 109

A.7. Movimientos de las hormigas. 110

Índice de cuadros

4.1. Problemas Reales. 49

4.2. Codificación mediante array de dos dimensiones. 51

4.3. Codificación mediante array de una dimensión. 51

4.4. Pesos asociados a las restricciones y preferencias. 54

4.5. Codificación de la Solución en un Cromosoma. 55

4.6. Mejores costos obtenidos (ejecuciones de 30' realizadas en la calibración). 70

5.1. Parámetros utilizados con Algoritmos Genéticos. 75

5.2. Parámetros utilizados con Búsqueda Tabú. 75

5.3. Parámetros utilizados con Ant System. 76

5.4. Parámetros utilizados con Ant Colony System. 76

5.5. Parámetros utilizados con $\mathcal{MAX} - \mathcal{MIN}$ Ant System. 76

5.6. Mejores Costos obtenidos. 77

5.7. Mejores Costos obtenidos 20032-5s/P2 vs. 20041-5s/P2. 82

Índice de Algoritmos

1. *Metaheurística GA* 58
2. *InicializarPoblación*($tipo_{pi}, tamano_p$) 58
3. *ReproducciónPoblación*($P, tipo_r, probabilidad_t$) 58
4. *CruzamientoPoblación*($Q, puntos_c, probabilidad_c$) 58
5. *MutaciónPoblación*($Q, probabilidad_m$) 58
6. *Metaheurística TS* 61
7. *Metaheurística AS* 64
8. *GeneraciónActividadHormigas()* 64
9. *NuevaHormiga(k)* 64
10. *EvaporaciónFeromona()* 64
11. *Metaheurística ACS* 65
12. *GeneraciónActividadHormigas()* 65
13. *NuevaHormiga(k)* 66
14. *AccionesGlobales()* 66
15. *Metaheurística* $\mathcal{MM}AS$ 67
16. *GeneraciónActividadHormigas()* 67
17. *NuevaHormiga(k)* 67
18. *AccionesGlobales()* 68
19. *SoluciónInicial* 69
20. *Metaheurística GA* 100
21. *Metaheurística TS* 107
22. *Metaheurística ACO* 116
23. *GeneraciónHormigasActividad()* 116
24. *NuevaHormiga(k)* 117
25. *NuevaHormiga(k)* 120
26. *EvaporaciónFeromona()* 120
27. *NuevaHormiga(k)* 122
28. *AccionesGlobales()* 123
29. *AccionesGlobales()* 125

Lista de Abreviaturas y Símbolos

ACO	Ant Colony Optimization - Optimización por Colonia de Hormigas
ACS	Ant Colony System
AS	Ant System
ATP	Assignment-Type Problem - Problema de Asignación
CLP	Constraint Logic Programming - Programación Lógica de Restricciones
CSP	Constraint Satisfaction Problem - Problema de Satisfacción de Restricciones
DM	Descent Methods - Métodos de la Pendiente
ECRC	European Computer-Industry Research Centre - http://www.ecrc.de
ES	Expert Systems - Sistemas Expertos
GA	Algoritmos Genéticos - Genetic Algorithms
GATP	General Assignment-Type Problem - Problema General de Asignación
GCP	Graph Colouring Problem - Problema de Coloreo de Grafo
GSP	Grouping Subproblem o Student Sectioning - Subproblema de Agrupación
$\mathcal{MM}$AS	$\mathcal{MAX} - \mathcal{MIN}$ Ant System
MA	Algoritmos Meméticos - Memetic Algorithms
NN	Neural Nets - Redes Neuronales
PASHA	Problema de Asignación de Salones y Horarios a Asignaturas
PATAT	Practice And Theory of Automated Timetabling http://www.asap.cs.nott.ac.uk/patat/patat-index.shtml
QAP	Quadratic Assignment Problem - Problema de Asignación Cuadrática
SA	Simulated Annealing
SP	Scheduling Problem - Problema de Programación
TS	Búsqueda Tabú - Tabu Search
TSP	Traveling Salesman Problem - Problema del Viajante de Comercio
TTP	Timetabling Problem - Programación de Horarios
UCTTP	University Course Timetabling Problem - Problema de Programación de Horarios de Cursos de una Universidad
UETTP	University Examination Timetabling Problem - Problema de Programación de Horarios de Exámenes de una Universidad
WATT	The Working Group on Automated Timetabling http://www.asap.cs.nott.ac.uk/watt/index.html

Capítulo 1

Introducción

1.1. Contexto

Desde siempre casi todas las actividades del hombre se rigen en función de horarios prefijados de antemano. Si se tiene que ir a trabajar, tomar un omnibus, asistir a una conferencia, tomar o brindar una clase o examen, previamente se debe saber el horario fijado para cada una de esas actividades.

¿Existe un problema en común detrás de todas estas actividades de modo que su solución diga lo qué hacer y cuándo? Si, efectivamente existe un problema en común y se llama Problema de Programación (Scheduling Problem, SP) [Wre96]. El SP general se describe como el proceso de seleccionar y asignar recursos y tiempo al conjunto de actividades de una planificación, y teniendo en cuenta que la asignación debe cumplir con un conjunto de restricciones que reflejan la relación temporal entre las actividades y la capacidad limitada de los recursos compartidos [Wre96]. Una forma particular del SP es el Problema de Programación de Horarios (Timetabling Problem) [Wre96].

Este tipo de problemas, generalmente son problemas de optimización difíciles de resolver en forma exacta, en donde la complejidad de los métodos exactos aumenta de manera exponencial según el tamaño del problema y hace que la aplicación de los mismos sea inviable dado la gran cantidad de recursos que son necesarios para solucionarlos, principalmente en tiempo de proceso.

Cuando los métodos exactos, que son métodos de búsqueda exhaustiva o enumerativa, no son aplicables aparecen los métodos heurísticos como una alternativa para solucionar estos problemas. Aunque los métodos heurísticos no pueden garantizar que encuentren las soluciones óptimas y tan siquiera que obtengan soluciones cercanas a las óptimas, la mayoría de ellos se comportan de forma satisfactoria obteniendo soluciones aceptables con un consumo aceptable de los recursos disponibles.

Las Heurísticas son métodos que buscan soluciones buenas (cercanas al óptimo) en un tiempo razonable, pero sin garantizar la factibilidad u optimalidad de las soluciones y en muchos casos sin establecer cuan cerca del óptimo se haya la solución factible encontrada. En general las heurísticas no recorren todo el espacio de soluciones, obteniendo solo soluciones localmente óptimas.

En un nivel superior de abstracción existen las Metaheurísticas [CDM+96] que pueden definirse como procesos iterativos que guían a otras heurísticas combinando distintos aspectos para explorar y explotar el espacio de búsqueda usando estrategias de aprendizaje. Surgen derivadas de la observación de los procesos que ocurren en los fenómenos físicos, biológicos y sociales de la naturaleza, extrayendo de esos fenómenos ciertas características que pueden ser aplicadas en la resolución de los problemas de optimización. Una de las características que tienen las metaheurísticas es que han sido definidas de manera de que sean robustas (aplicables a una gran variedad de problemas con una mínima, sí fuera necesaria, modificación de su definición básica) y efectivas como herramientas de optimización, las cuales comienzan con una solución inicial o una población de individuos iniciales y se van modificando iterativamente hasta que se llegue a la condición de fin, la cual se puede definir como haber obtenido una solución considerada aceptable o bien llegar al número máximo de iteraciones.

1.2. Motivación

En la Facultad de Ingeniería, semestre a semestre, se encuentran con el problema de asignarle a cada curso de las diferentes carreras un salón y un horario específico. Este problema es un problema de programación de horarios de difícil solución dada la escasez de recursos que tiene la Facultad, principalmente la disponibilidad de salones.

Originalmente el problema se resolvía en forma manual por parte de Bedelía, trabajo que insumía varias semanas. Posteriormente se desarrolló un sistema de información de apoyo a la decisión, y se implementó como método de solución aproximado la metaheurística Búsqueda Tabú [CPU03]. A partir de datos existentes, y una solución inicial, interactivamente el usuario obtiene una solución aproximada, que puede ajustar según su propio sentido común y conocimiento de la realidad en la que actúa. Otro trabajo anterior [RC99] realizó un estudio sobre metaheurísticas para el problema de asignación e implementó Ant System pero de forma independiente y sin los datos de la aplicación real.

1.3. Objetivos

Esta tesis se plantea como objetivo principal la adaptación, implementación y aplicación de un conjunto de metaheurísticas para el problema planteado.

Los principales objetivos son:

- Realizar una actualización del estado del arte en cuanto al Problema de Programación de Horarios.
- Construir un motor de metaheurísticas.
- Realizar un estudio sobre calibrado y comparación de resultados de las metaheurísticas implementadas.

1.4. Limitaciones de las Soluciones existentes

Dada la característica del tipo de problema es casi imposible llegar a una definición y formalización del mismo que sea común a todas las instituciones universitarias del mundo. Cada una de ellas tiene su característica particular en la asignación de los cursos. Algunos intentos se han realizado procurando llegar a una formalización común, pero cada institución e investigador que tiene que resolver el problema debe tener en cuenta las particularidades de la institución a la que pertenece. En consecuencia, la Facultad de Ingeniería no es una excepción a la realidad planteada, se están realizando esfuerzos en construir un marco que permita resolver el problema particular de esta institución.

Dos trabajos anteriores utilizaron dos métodos para resolver el problema de asignación de salones, pero cada uno realizado en forma separada y utilizando una definición diferente del mismo problema [RC99], [CPU03].

1.5. Contribución Principal

En este trabajo se realiza una búsqueda bibliográfica que permite poner a punto el estado del arte para los problemas de programación de horarios. Se estudian algunas de las metaheurísticas mas importantes como Algoritmos Genéticos, Búsqueda Tabú y dentro de Optimización por Colonia de Hormigas los algoritmos Ant System, Ant Colony System y $\mathcal{MAX} - \mathcal{MIN}$ Ant System. Se realizan las adaptaciones para cada una de ellas al problema particular de la Facultad. Finalmente se realiza la

adaptación, implementación y comparación de las mismas para diferentes instancias reales del problema.

1.6. Organización

El siguiente informe está organizado de la siguiente manera, en el capítulo 2, se presenta una puesta a punto de la realidad en el mundo referente a problemas de programación de horarios, mostrando diferentes tipos de problemas, definiciones y métodos de resolución. En el capítulo 3 la definición del problema de programación de horarios particular de la Facultad y su formalización, en el capítulo 4 se presenta la adaptación de las metaheurísticas al problema puntual de la Facultad, finalmente en el capítulo 5 se muestran los resultados obtenidos y en el capítulo 6 las conclusiones del presente trabajo. En el apéndice A se presenta la descripción de las diferentes metaheurísticas utilizadas (Ant System, Ant Colony System, $\mathcal{MAX} - \mathcal{MIN}$ Ant System, Algoritmos Genéticos y Búsqueda Tabú) y en el apéndice B se presenta la calibración de los parámetros.

Capítulo 2

Estado del Arte

2.1. Introducción

Si se observa cuidadosamente la mayoría de las actividades que se desarrollan a diario durante el desarrollo de la vida de las personas, se puede afirmar que existe un horario determinado con anterioridad que dice en que horario se deben realizar cada una de esas actividades. Si se tiene que ir al trabajo o a estudiar, existen horarios definidos en los que pasan los ómnibus. En la universidad, si se tiene que asistir a ciertas clases en ciertos horarios. Si se necesita viajar en avión, el vuelo es programado para salir a una hora específica. En la oficina se tiene la obligación de asistir a las reuniones en el horario indicado. En las fábricas, ciertos productos se deben producir para una fecha dada usando diferentes máquinas a través de procesos de fabricación. En el hospital, las enfermeras son asignadas de manera de cubrir todo el día para poder atender las necesidades de los pacientes.

¿Hay un problema común detrás en todas estas actividades cuya solución determine qué hacer y cuándo? Si, de hecho ese problema existe y se llama Problema de Programación (Scheduling Problem, SP). El SP general se describe como el proceso de seleccionar y asignar recursos y tiempo al conjunto de actividades de una planificación, y teniendo en cuenta que la asignación debe cumplir con un conjunto de restricciones que reflejan la relación temporal entre las actividades y la capacidad limitada de los recursos compartidos [Wre96].

SP es definido por Anthony Wren [Wre96] como sigue: *"Scheduling may be seen as the arrangement of objects into a pattern in time or space in such a way that some goals are achieved, or nearly achieved, and that constraints on the way objects may be arranged are satisfied or nearly satisfied."*

El Problema de Programación de Horarios (Timetabling Problem) es una forma particular del SP. Wren [Wre96] hace distinción entre estos dos problemas definiendo

el Problema de Programación de Horarios (Timetabling Problem, TTP) como sigue: *"Timetable shows when particular events are to take place. It does not necessarily imply an allocation of resources."* Esto esencialmente significa que para el TTP es mas importante resolver cuándo los eventos van a ocurrir que en dónde van a ocurrir.

Existe el consenso en la comunidad científica que el TTP es de difícil generalización, debido a la diversidad de regímenes de educación y de las características de las instituciones, que varían de región en región. De esta forma, los sistemas son comúnmente desarrollados para atender una institución específica. Por ese motivo no existe un conjunto de problemas que puedan ser usados para validar los diferentes algoritmos desarrollados, aunque algunos esfuerzos en ese sentido se están desarrollando (The Working Group on Automated Timetabling - WATT[a]).

Una investigación interesante sobre cuestiones relacionadas con el proceso aplicado en la construcción de la programación de los horarios en escuelas secundarias [KKK97] revela que el 80 % de las instituciones en Hong Kong usan computadoras para realizar la programación de los horarios. Entretanto, apenas un tercio de ellas usan programas específicos. Las demás escuelas realizan la programación en forma manual o utilizan las computadoras para verificar la existencia de conflictos.

En la Facultad se están realizando esfuerzos en automatizar el TTP particular de la institución. Dos trabajos anteriores utilizaron dos métodos Metaheurísticos para resolver el problema [RC99], [CPU03].

Muchos autores, entre ellos Schaerf [Sch95] y de Werra [dW85], creen que el TTP no puede ser completamente automatizado. Las razones dadas por ellos son dos: por un lado, hay razones que indican que hacer un horario mejor que otro no puede ser fácilmente expresado en un sistema automatizado y por otro lado, desde que el espacio de búsqueda es usualmente enorme, la intervención humana puede guiar la búsqueda hacia prometedoras direcciones que el sistema por si solo difícilmente sea capaz de encontrar. Por las razones anteriores, la mayoría de los sistemas permiten al usuario ajustar manualmente la solución final. Sin embargo, algunos sistemas requieren bastante intervención humana, de modo que esos se llaman sistemas de programación de horarios interactivos o semi-automáticos.

Los sistemas descritos en [CdW89], [DMV89], [WN90], [MC91] son ejemplos de sistemas interactivos.

La literatura sobre TTP es muy amplia y variada. Schmidt y Strohlein [SS79] proveen referencias bibliográficas incluyendo mas de 200 artículos aparecidos antes de 1979.

Junginger [Jun86] describe las investigaciones sobre el TTP para escuelas alemanas. En particular, describe los diferentes programas implementados y su utilización por

[a]WATT - http://www.asap.cs.nott.ac.uk/watt/index.html

las instituciones.

De Werra [dW85] describe formalmente varios problemas y proporciona sus respectivas formulaciones. También describe las investigaciones mas importantes aplicando teoría de grafos.

Carter [Car86] examina el Problema de Programación de Horarios de Exámenes de una Universidad (University Examination Timetabling Problem, UETTP) y se centra principalmente en la reducción al Problema de Coloreo de Grafo (Graph Colouring Problem, GCP).

Corne, Roos y Fang [CRF94] proporcionan trabajos que aplican algoritmos genéticos al TTP, discuten también las perspectivas futuras y comparan los resultados obtenidos con respecto a algunas otras investigaciones. Otras investigaciones se pueden encontrar en [DLU73], [Hil81], [Kle83], [Vin84] y [FRL86].

2.1.1. El Problema de Programación de Horarios

Un TTP consiste en programar varios eventos (exámenes, cursos, reuniones, etc.) a un limitado número de períodos de tiempo satisfaciendo un conjunto de restricciones para esa asignación. Esas restricciones se dividen en Restricciones Rígidas (Duras, Hard) y Restricciones Flexibles (Blandas, Soft) [BKJW97].

La asignación de recursos es a menudo un aspecto importante a tomar en cuenta mientras se crea una programación de horarios. Wren define el TTP como sigue [Wre96]: *"Timetabling is the allocation, subject to constraints, of given resources to objects being placed in space time, in such a way as to satisfy as nearly as possible a set of desirable objectives."*

Las restricciones flexibles, también llamadas Preferencias, tienen un costo asociado y si alguna de ellas no es satisfecha la meta es minimizar su costo, mientras que las restricciones rígidas deben ser satisfechas, de modo que su costo asociado debe ser cero.

Una asignación de horarios es factible si satisface todas las restricciones rígidas. Las restricciones rígidas generalmente tienen que ver con las restricciones de infraestructura edilicia, de recursos materiales disponibles y referentes a restricciones de los profesores y los alumnos. Esto incluye los eventos que no pueden solaparse en el tiempo, por ejemplo:

- Clases dadas por el mismo profesor.
- Clases programadas en el mismo salón.

Otros ejemplos de restricciones rígidas:

- Una clase no puede ser asignada a un salón particular de menor capacidad que la cantidad de alumnos de la clase.
- Los alumnos y los profesores no pueden ser asignados en más de un lugar a la vez.
- Para cada período de tiempo es necesario que existan suficientes recursos disponibles (salones, laboratorios, salones especiales, etc.).

Las restricciones flexibles son las preferencias que no se ocupan de los conflictos de tiempo y tienen un costo más bajo asociadas a ellas. En los problemas reales usualmente es imposible satisfacer todas las restricciones flexibles, de modo que se intenta reducir el costo pero no se espera poder reducirlo a cero.

Son ejemplos:

- Un curso o examen necesita ser asignado en un determinado período de tiempo.
- Un curso o examen debe ser programado antes o después de otro.
- Distribuir las diferentes clases de un curso durante la semana, por ejemplo (Lunes, Miércoles y Viernes) o (Martes y Jueves).
- Los profesores prefieren tener días libres y tener la mayoría de las clases en forma consecutiva en algunos días.
- Los profesores tienen preferencias por algunos salones o tipo de salones.

Algunas restricciones flexibles pueden tener un costo más alto que otras. Por ejemplo, las preferencias que implican a los profesores pueden tener mayor prioridad que las preferencias de los estudiantes.

La función de costo mide la calidad de la solución e implica la suma de los costos de las restricciones (rígidas y flexibles) que son violadas. El objetivo de cualquier técnica de optimización es reducir al mínimo posible la función de costo.

Los TTP son los que generan mayor trabajo administrativo para una gran variedad de instituciones. La construcción de la solución manual es una tarea compleja y normalmente requiere varios días de trabajo. La construcción de un cuadro de horarios por esta vía puede demandar varias semanas en una escuela secundaria [Bar96], [KKK97] o hasta meses en una universidad [Car86]. Adicionalmente, la solución obtenida puede ser insatisfactoria en diferente aspectos, por ejemplo, un estudiante puede verse impedido de tomar un curso que quiera porque el mismo está en conflicto con otro o

un profesor puede quedar descontento al tener muchos horas puentes. Por esta razón anterior, el problema ha requerido gran atención en los últimos cuarenta años. Los primeros esfuerzos en este sentido comenzaron en la década del 60 con los trabajos precursores de Gotlieb y Csima en [CG61] y [Got63]. Desde entonces muchos artículos relacionados con la automatización del TTP aparecieron en diferentes conferencias y revistas científicas. Según Bardadym [Bar96] hasta el año 1995 existían mas de 700 artículos relacionados con el tema. En 1995 se realizó la primera conferencia internacional dedicada exclusivamente a este tema (Practice And Theory of Automated Timetabling - PATAT[b]), luego en 1997 la segunda, la tercera en 2000, la cuarta en 2002 y la última en 2004.

2.1.2. Tipos de Problemas de Programación de Horarios

Mientras todos los TTP comparten las mismas características básicas del problema general, la gran cantidad de variantes entre los diferentes problemas hacen que sea prácticamente imposible construir soluciones generales para todos los problemas. Debido a esto la mayoría de las investigaciones se especializan en un problema concreto o en un subconjunto de problemas con características similares. Este acercamiento permite que los investigadores aprovechen la naturaleza del problema y utilicen el conocimiento sobre el espacio de la solución para implementar los métodos que pueden producir asignaciones de horarios de buena calidad. En la sección 2.2 se presenta la formulación matemática del problema.

- Problema de Programación de Horarios del Transporte Público
  (Public Transport Timetabling Problem) [Wre81] es quizás el mas común de los TTP. El problema aplica al transporte de personas en los diferentes medios (ómnibus, tren, avión, etc.) El problema difiere de todos los otros en que los recursos son móviles y recorren diferentes distancias para completar sus tareas. Para el caso del problema de la universidad los estudiantes deben desplazarse entre los lugares en donde se dictan las clases, pero este aspecto generalmente no forma parte del problema. Sin embargo, para el problema del transporte el desplazamiento de los recursos es indispensable y parte importante del problema.

- Problema de Programación de Horarios de Empleados
  (Employee Timetabling Problem) [MGS96] describe el problema donde los empleados de una organización tienen habilidades especiales y la organización trabaja en diferentes rangos de horario, por ejemplo la operación de un hospital. Debido a la naturaleza del trabajo de enfermería el servicio debe ser proporcionado siempre. Generalmente cada día esta repartido en tres períodos de ocho

[b]PATAT - http://www.asap.cs.nott.ac.uk/patat/patat-index.shtml

horas cada uno. El problema es asignar las suficientes enfermeras a cada turno de manera de cumplir con todas las restricciones prácticas.

- Problema de Programación de Horarios de Escuelas Secundarias
(School Timetabling Problem) es un problema difícil que enfrentan las escuelas. Generalmente también se le llama Problema de Asignación de Salones-Profesores (Class-Teacher Room Assignment Problem). A cada clase se le debe asignar un profesor y un salón para cada período de tiempo disponible en la semana. Aparte de producir un horario factible es necesario balancear las cargas del profesor, por ejemplo se requiere a menudo para cada profesor tener una mañana o una tarde libre una vez por semana.

- Problema de Programación de Horarios de Universidades
(University Timetabling Problem, UTTP) se pueden separar en dos: Problema de Programación de Horarios de Cursos de una Universidad (University Course Timetabling Problem, UCTTP) [CL98] y Problema de Programación de Horarios de Exámenes de una Universidad (University Examination Timetabling Problem, UETTP) [CL96], [BF97]. El UCTTP es el problema de resolver la programación semanal de todas las clases de los cursos de la universidad, minimizando la superposición de las clases que tienen alumnos en común. El UETTP resuelve la programación de los exámenes de la universidad evitando la superposición de exámenes con alumnos en común.

2.1.3. Problema de Búsqueda y de Optimización

Cada TTP puede ser clasificado en dos tipos, dependiendo de las necesidad de optimizar o no la función objetivo:

- Problema de Búsqueda, en este caso el problema consiste en encontrar una programación de horarios que sea viable, o sea que satisfaga todas las restricciones rígidas.

- Problema de Optimización, en este caso el problema consiste en encontrar una programación de horarios que satisfaga las restricciones rígidas y minimice (o maximice) la función objetivo que incluye las restricciones flexibles.

En la sección 2.2.1 se describe la formulación del UCTTP como un Problema de Búsqueda y en la sección 2.2.2 como un Problema de Optimización, ambas formulaciones dadas por de Werra [dW85].

2.1.4. Complejidad

Dos de las principales características del TTP son la gran complejidad del problema y la cantidad de variaciones del mismo que existen en la realidad. En [EIS76], Even, Itai y Shamir demostraron que los TTP mas comunes son $\mathcal{NP}$-completos aplicando una reducción del GCP. Debido a las diferentes variaciones existentes del problema, un modelo general para el TTP no existe y aunque generalmente para esas diferentes variaciones del TTP se pruebe que son $\mathcal{NP}$-completos, la simple eliminación de una sola restricción puede hacer que el problema pase a ser resuelto en forma polinomial.

Cooper y Kingston [CK96] demostraron que una serie de problemas de horarios que aparecen comúnmente en la práctica son $\mathcal{NP}$-completos. Específicamente, los autores mostraron que la $\mathcal{NP}$-completitud aparece cuando cada estudiantes puede elegir entre un conjunto de asignaturas, una de las características principales de los UCTTP, y cuando las clases son de diferente duración.

Algunas variaciones del TTP pueden ser resueltas en tiempo polinomial. Para el caso del problema de programación de horarios de escuelas secundarias, en su versión básica, cuando los profesores y grupos están siempre disponibles, Even, Itai y Shamir [EIS76] presentaron una solución mediante un método basado en flujo de grafos. Algunas condiciones para la existencia de una solución viable para el problema básico de programación de horarios de escuelas secundarias, incluyendo restricciones de los profesores y de los grupos, son establecidas por de Werra [dW96], [dW97]. de Werra presenta, también, algunas variantes que pueden ser resueltas en tiempo polinomial.

La mayoría de los casos resueltos en tiempo polinomial, no incluyen las restricciones mas comunes que aparecen en los problemas reales. En virtud de los anterior, se justifica el encare del problema mediante técnicas heurísticas, las cuales no garantizan la existencia de una solución viable y tampoco su optimalidad [Pea84].

2.1.5. Acercamiento de la Solución

La mayoría de las primeras técnicas [SS79] fueron basadas en la simulación de la forma humana de resolver los problemas. Estas técnicas, llamadas Heurísticas Constructivas, consisten en la construcción gradual del cuadro de horarios, agregando clase por clase. La idea principal es asignar las clases más difíciles primero y se diferencian entre si en el significado que le dan a la expresión “más difíciles”. Un ejemplo típico de ésta técnica se puede ver en [Jun86].

Mas tarde, los investigadores comenzaron a aplicar técnicas generales, como por ejemplo programación entera, flujo en redes y otros. También el problema ha sido estudiado reduciéndolo al muy estudiado GCP.

Mas recientemente se utilizan métodos basados en técnicas de búsqueda, como por ejemplo Simulated Annealing, Búsqueda Tabú, Algoritmos Genéticos, etc.

2.2. Formulación del UCTTP

Los TTP para los cursos de una universidad consisten en programar un conjunto de clases para cada curso dentro de un número dado de salones y períodos de tiempo. La diferencia principal con el problema de las escuelas secundarias es que los cursos de la universidad pueden tener estudiantes comunes, mientras que las clases de la escuela secundaria son con diferentes estudiantes. Si dos cursos tienen estudiantes comunes entonces están en conflicto y no se pueden programar en el mismo período. Por otra parte, los profesores de la escuela enseñan siempre a más de una clase, mientras que en universidades, por lo general un profesor suele enseñar solamente un curso. Además, en el problema de la universidad, la disponibilidad de los salones y su capacidad desempeña un papel importante, mientras que en el problema de la escuela secundaria la disponibilidad y capacidad de los salones, en la mayoría de los casos, no es tenido en cuenta ya que se asume generalmente que cada clase tiene su propio salón asignado.

En esta sección se describe dos formulaciones diferentes del UCTTP. Primero la formalización como un Problema Básico de Búsqueda y luego se introduce el Problema de Optimización y se discute las diferentes variantes del problema. Finalmente la formulación del problema como un Problema General de Asignación.

2.2.1. Problema Básico de Búsqueda

Existen varias formulaciones del UCTTP en [Tri92]. En [dW85] se define como un Problema Básico de Búsqueda (Basic Search Problem) de la siguiente forma:

Sean q cursos $K_1, ..., K_q$, y para cada $i = 1, ..., q$, el curso K_i está formado por k_i clases. Sean r planes de estudio $S_1, ..., S_r$, grupo de cursos que tienen estudiantes en común. Esto indica que los cursos en S_l deben ser programados todos en diferentes horarios. El número de períodos de tiempo es p y para cada $j = 1, ..., p$, l_j es el máximo número de clases que pueden ser programadas en el período j (número de salones disponibles en el período j).

En esta definición, de Werra considera que todas las clases tienen la misma duración (un período de tiempo).

La formalización es la siguiente:

$$buscar \quad y_{ij} \qquad (i = 1, ..., q; j = 1, ..., p)$$

$$s.a. \sum_{j=1}^{p} y_{ij} = k_i \qquad (i = 1, ..., q) \tag{2.1}$$

$$\sum_{i=1}^{q} y_{ij} \leq l_j \qquad (j = 1, ..., p) \tag{2.2}$$

$$\sum_{i \in S_l} y_{ij} \leq 1 \qquad (l = 1, ..., r; j = 1, ..., p) \tag{2.3}$$

$$y_{ij} = \{0, 1\} \qquad (i = 1, ..., q; j = 1, ..., p) \tag{2.4}$$

donde $y_{ij} = 1$ si una clase del curso K_i es programada en el período j, y $y_{ij} = 0$ sino.

La restricción (2.1) impone que cada curso tenga programado el número correcto de clases. La restricción (2.2) fuerza que para cada período de tiempo no haya mas clases asignadas que salones disponibles. La restricción (2.3) previene que las clases que deben ser programadas en diferentes horarios sean asignadas en el mismo período.

Se puede demostrar que el problema es $\mathcal{NP}$-completo a partir de una simple reducción del GCP.

2.2.2. Problema de Optimización

En [dW85] se plantea el TTP como un Problema de Optimización incluyendo la siguiente función objetivo:

$$max \sum_{i=1}^{q} \sum_{j=1}^{p} d_{ij} y_{ij}$$

donde d_{ij} es el beneficio de asignar una clase del curso K_i en el período j.

Triphaty en [Tri92] considera una matriz entera de conflictos $C_{q \times q}$, donde $c_{ii'}$ representa el número de estudiantes comunes que toman el curso K_i y el curso $K_{i'}$. De este modo, $c_{ii'}$ representa en alguna medida la insatisfacción en caso de asignar una clase de K_i y una clase de $K_{i'}$ en el mismo horario. Tripathy trata de minimizar la insatisfacción global.

Muchos autores, siguiendo [EL87], dividen los requerimientos entre duros y blandos. Los requerimientos duros son incluidos en las restricciones y definen el espacio de búsqueda, mientras los requerimientos blandos son incluidos en la función objetivo [AF89].

2.2.3. Variantes del Problema

A continuación se muestra algunas de las variantes mas comunes del problema básico.

Sesiones Múltiples y Subproblema de Agrupación

En ciertas universidades, algunos cursos se repiten mas de una vez durante la semana. En particular esos cursos involucran una gran cantidad de estudiantes y pertenecen a varios planes de estudios de modo que los mismos son dados en múltiples sesiones. La creación de diferentes sesiones para un curso puede ayudar a reducir el número de conflictos en un horario. Por ejemplo, el plan de estudio S_1 involucra los cursos K_1 y K_2 y el S_2 involucra los cursos K_1 y K_3. También una clase de K_2 tiene lugar en el tiempo p y una clase de K_3 en el tiempo q. En este caso, las clases del curso K_1 no pueden ser programados en el tiempo p y ni en q. Sin embargo, si el curso K_1 es dado en dos sesiones, entonces la clase de una sesión puede ser dada en el tiempo p, y la clase de la otra sesión en el tiempo q.

Dado un cierto programa de horarios, el problema de asignar los estudiantes que toman una plan de estudio particular a una sesión específica de un curso de manera de minimizar los conflictos es llamado Subproblema de Agrupación (Grouping Subproblem o Student Sectioning).

El Subproblema de Agrupación (Grouping Subproblem o Student Sectioning, GSP) es considerado, entre otros, en [LD86], [AF89], [Her91] y [Tri92]. Laporte y Desroches formulan el problema como un problema de optimización separando los requerimientos en duros y blandos. Los requerimientos duros son: (i) un estudiante no puede tener dos clases en el mismo tiempo y (ii) un estudiante no puede tener dos clases adyacentes en diferentes Campus[c]. Los requerimientos blandos incluyen el número de clases en un día común y cambios de Campus en las horas libres. El problema es resuelto en dos fases: en la primera, el algoritmo busca una solución factible, mientras que en la segunda busca un óptimo local.

En estas investigaciones que consideran el GSP, los algoritmos se basan primero en solucionar el TTP puro y luego solucionar el GSP. Por ejemplo, Hertz [Her91] utiliza un procedimiento iterativo que en cada iteración soluciona ambos problemas, primero

[c]Campus: edificios y terrenos de una universidad o de un colegio.

el TTP y después el GSP.

Períodos de largo variable

Hasta ahora se han considerado clases de un único período de duración. Muchos autores consideran también clases de diferente duración.

Las clases pueden durar varios y diferentes períodos de tiempo. Por ejemplo, en [FR85] las clases pueden durar uno, dos o tres períodos de tiempo.

De modo, que para cada clase se debe considerar el tiempo de inicio y la duración. En este caso, las restricciones (2.1) y (2.2) son reemplazadas por restricciones que tengan en cuenta el hecho de que dos clases l_i y l_j, comiencen en el tiempo p y $q > p$, están en conflicto si $q - p < d_{l_i}$, donde d_{l_i} es la duración de la clase l_i.

Hertz [Her91] trata la situación mas general en donde las clases pueden tener duración variable. En particular, Hertz especifica que un curso está compuesto de cierto número de períodos y que a su vez pueden darse en un número variable de clases de diferente duración. Por ejemplo, un curso integrado por doce períodos puede consistir en clases de dos o tres períodos. En este caso el curso se da en cuatro, cinco o seis clases de las siguientes longitudes: (3,3,3,3), (2,2,2,3,3) o (2,2,2,2,2,2).

Subproblema de Asignación de Salones de Clases

El Subproblema de Asignación de Salones de Clases (Classroom Assignment Subproblem) consiste en asignar las clases a los salones, dada un programación de horario fijo. Carter y Tovey [CT89] analizan en detalle este problema y dan varias formulaciones alternativas y variantes. Ellos también muestran en qué caso el problema es $\mathcal{NP}$-completo. En particular, muestran que el problema es $\mathcal{NP}$-completo cuando se impone la restricción de que todas las clases de un curso deben ser dadas en el mismo salón, sin embargo, el problema es polinomial si se considera la asignación en cada período independientemente de otro período.

Ferland y Roy [FR85] resuelven el Subproblema de Asignación de Salones de Clases reduciéndolo al Problema de Asignación Cuadrática (Quadratic Assignment Problem, QAP).

2.2.4. Problema General de Asignación

La noción del Problema de Asignación (Assignment-Type Problem, ATP) y su versión general Problema General de Asignación (General Assignment-Type Problem, GATP)

son utilizados bastante como herramienta de modelado de problemas. En particular, esos modelos son muy apropiados para la formulación del TTP y del SP donde los períodos de tiempo tienen que ser determinados para las actividades de acuerdo a las restricciones especificadas. En esta sección se presenta el ATP, el GATP y de como el UCTTP puede ser formulado como un GATP. Las siguientes formulaciones son dadas por Ferland en [Fer98].

Problema de Asignación

El ATP se define como sigue [FHL96] [FL92]:

Dado n items y m recursos, el problema es el de determinar una asignación de cada item a los recursos de manera de optimizar una función objetivo que satisfaga K restricciones.

El modelo matemático asociado al ATP es formulado como sigue:

$$\begin{aligned} & Min\ F(x) \\ s.a.\ & G_k(x) \leq 0 \qquad 1 \leq k \leq K \\ & x \in X(1) \end{aligned}$$

donde $X(1) = \{x : \sum_{j \in J_i} x_{ij} = 1, 1 \leq i \leq n; x_{ij} = \{0,1\}, 1 \leq i \leq n, j \in J_i\}$, y $J_i \subset \{1, 2, ..., m\}$ es el conjunto de recursos admisibles para el item $i, 1 \leq i \leq n$. La variable de decisión x_{ij} está dada por:

$$x_{ij} = \begin{cases} 1 & \text{si el item } i \text{ es asignado al recurso } j \\ 0 & \text{sino.} \end{cases}$$

La función objetivo F y el lado izquierdo de la restricción $G_k, 1 \leq k \leq K$, solo se requiere que sean calculables.

Problema General de Asignación

Un GATP [Fer98] es un ATP donde cada item i es asignado a a_i recursos, $a_i \geq 1$. El modelo matemático asociado al GATP es:

$$\begin{aligned} & Min\ F(x) \\ s.a.\ & G_k(x) \leq 0 \qquad 1 \leq k \leq K \end{aligned}$$

$$x \in X(a)$$

donde $X(a) = \{x : \sum_{j \in J_i} x_{ij} = a_{ij}, 1 \leq i \leq n; x_{ij} = \{0,1\}, 1 \leq i \leq n, j \in J_i\}, a = [a_1, a_2, ..., a_n]^T$, y $a_i \geq 1$ y entera $1 \leq i \leq n$.

Un ATP es un GATP donde $a = [1, 1, ..., 1]^T$.

Problema de Programación de Horarios de Cursos de una Universidad como Problema General de Asignación

Para formular el UCTTP como un GATP, las clases son los items y el tiempo de comienzo de las clases son los recursos. Entonces para cada clase $i, 1 \leq i \leq n$, tiempo de comienzo $j \in J_i$ (el conjunto de tiempos de comienzo admisibles para la clase i)

$$x_{ij} = \begin{cases} 1 & \text{si la clase } i \text{ comienza en el tiempo } j \\ 0 & \text{sino.} \end{cases}$$

La función objetivo incluye las preferencias de los alumnos:

$$F(x) = \sum_{i=1}^{n} \sum_{j \in J_i} c_{ij} x_{ij}$$

donde c_{ij} es el costo de comenzar la clase i en el tiempo j especificado en términos de preferencias de los alumnos.

La formulación matemática del problema es:

$$\begin{aligned} & Min \sum_{i=1}^{n} \sum_{j \in J_i} c_{ij} x_{ij} \\ s.a.\ & x_{ij}.x_{kl} \leq 0 && (k,l) \in \Gamma_{ij}, 1 \leq i \leq n, j \in J_i \\ & \sum_{(i,j) \in K_{bt}} x_{ij} - U_b \leq 0 && 1 \leq b \leq B, 1 \leq t \leq T \\ & x \in X(1) \end{aligned}$$

donde la primera restricción es especificada para eliminar los conflictos en donde a los estudiantes se les asigna clases que son impartidas simultáneamente. Para esos se define $\Gamma_{ij} = \{(k,l) :$ clases i y k tienen estudiantes en común, y la superposición de tiempo si la clase i comienza en j y la clase k en $l\}$. La segunda restricción cuenta la disponibilidad de los salones. Dividiendo el conjunto de salones en diferentes subconjuntos incluyendo salones del mismo tipo. Se definen $B =$ número de tipos de salones y $U_b =$ número de salones del tipo $b, 1 \leq b \leq B$ y para cada clase un único tipo de salón es especificado, $K_{bt} = \{(i,j)$: clase i que requiere un salón de tipo b y ocurre durante la hora de enseñanza t de la semana siempre que el horario de comienzo sea $j\}$ y T es el número total de horas de clase de la semana.

2.3. Métodos y Técnicas de Solución

La construcción de una solución al TTP puede ser una tarea extremadamente dificultosa y la búsqueda de una solución manual requiere mucho esfuerzo. El problema ha atraído la atención de la comunidad científica de un gran número de disciplinas (incluida Investigación Operativa e Inteligencia Artificial) durante los últimos cuarenta años y en la última década ha aumentado el interés en ese campo.

Una gran variedad de trabajos para el TTP han sido descrito en la literatura y probados en casos reales.

2.3.1. Métodos Exactos

Programación Matemática

Muchos autores solucionan el TTP usando técnicas de programación entera, ver [Akk73], [Bre76], [SS77], [Tri80], [MW84], [Tri84], [Tri84], [FR85], [Tri92].

Por ejemplo, Tripathy [Tri80] utiliza una técnica de relajación lagrangeana para tratar un problema de ubicación de cursos que se reduce a un problema profesor-grupo básico, en [Tri84] utiliza programación entera y en [Tri92] extiende la técnica para tratar también el GSP. Ferland y Roy [FR85] formulan el problema como un Problema de Asignación que lo resuelven a través de una reducción al Problema de Asignación Cuadrática (Quadratic Assignment Problem, QAP).

Desafortunadamente la formulación matemática de los TTP genera un número importante de variables y restricciones, que hacen que, para instancias de tamaño real, el problema no sea tratable en forma práctica [Car86]. Sin embargo algunos intentos por reducir el tamaño de los problemas fueron propuestos en la literatura. Por ejemplo, Lennon [Len86] reduce el espacio de búsqueda empleando Programación Matemática solamente para programar los exámenes mas complicados y luego utiliza un método heurístico para programar el resto de los exámenes.

Programación Dinámica

La programación dinámica es un método de búsqueda enumerativo implícito, el cual puede ser visto como una técnica de Dividir & Conquistar. Para solucionar los TTP grandes, primero se descompone en varios subproblemas independientes mas pequeños. Para los TTP de tamaño real, la aplicación del método no es práctico, ver por ejemplo [HK62].

Branch & Bound

La búsqueda Branch & Bound es también un método enumerativo implícito. Consiste en dos procedimientos fundamentales: Branch es el proceso de repartir un problema grande en dos o más subproblemas y Bound es el proceso de calcular un límite más bajo en la solución óptima de un subproblema dado. Para problemas pequeños (algunas decenas de elementos) este método brinda una solución óptima. Sin embargo, si el tamaño del problema crece (para instancias de tamaño real de algunas centenas de elementos), el tamaño del espacio de soluciones crece exponencialmente y también el tiempo de ejecución del método. Ver [Bal69], [MF75], [BM85], [CP89].

Flujo en Redes

Osterman y de Werra [OdW83] redujeron el TTP a una secuencia de problemas de Flujo en Redes. Crearon una red para cada período de tiempo de modo que el flujo en la red identifica la clase dada en el período. de Werra [dW85] propuso un método similar creando una red por cada curso. La solución de la red da el horario para todas las clases del curso. La construcción de la red es repetida para todos los cursos y eventualmente, si una solución es encontrada para todas las redes, se completa la asignación de horarios para todos los cursos. Desde que no hay vuelta atrás en los cursos ya programados mientras se construye la solución no existe garantía de que la solución encontrada exista. Otros autores que utilizaron flujo en redes son [DM76], [CdW89] y [DMV89].

2.3.2. Métodos Secuenciales

Estos métodos ordenan los eventos (cursos, exámenes) usando heurísticas y los asignan secuencialmente en los períodos válidos de tiempo de modo que ningún evento en un período esté en conflicto con otro [Car86].

En los métodos secuenciales, el TTP es usualmente representado como un grafo en donde los eventos son representados por vértices, mientras que los conflictos entre los eventos son representados por aristas [dW85].

Problema de Coloreo de Grafo

Por ejemplo, si un estudiante tiene dos eventos, existe una arista entre los dos nodos representando el conflicto. La construcción de la programación de los horarios libre de conflictos puede ser modelada como un GCP. Cada período de tiempo corresponde con un color en el GCP y los vértices del grafo son coloreados de manera que dos vértices

adyacentes queden con el mismo color. El GCP es uno de los clásicos problemas $\mathcal{NP}$-completo [GJ79].

Una variedad de heurísticas de coloreo de grafos para la construcción de la solución al TTP sin conflictos se puede ver en [Bre79], [CL96]. Estas heurísticas ordenan los eventos basados en la estimación de cuan dificultoso es asignarle un horario. Las heurísticas que se utilizan a menudo son:

- Grado más Grande Primero (Largest degree first): Los eventos con mayor número de conflictos son programados primero. La razón es que los eventos con una gran cantidad de conflictos son más difíciles de programar de modo que hay que abordarlos primero.

- Grado más Pesado Primero (Largest weighted degree): es una modificación del anterior en donde el peso de cada conflicto está dado por la cantidad de estudiantes involucrados.

- Grado de Saturación (Saturation degree): En cada paso de la construcción de la solución se selecciona el evento que tenga menor número de períodos de tiempo disponibles teniendo en cuenta lo construido hasta ese momento.

- Grado del Color (Color degree): Esta heurística da la prioridad a los eventos que tengan el número mas grande de conflictos con los eventos que ya se hayan programado.

Una vez que los eventos son ordenados, se pueden emplear diferentes métodos para seleccionar un horario válido, por ejemplo, se puede seleccionar el primer período de tiempo válido o el "mejor" en el contexto de la función objetivo definida.

2.3.3. Métodos de Agrupamiento

En estos métodos los eventos son agrupados en grupos que satisfacen las restricciones rígidas (ningún evento está en conflicto con otro). Esto es un proceso muy similar al utilizado con el GCP. Luego se asignan los grupos de eventos a los períodos de tiempo de manera de satisfacer las restricciones flexibles. Un primer trabajo que describe esta idea fue escrito por White y Chan [WC79]. Diversas técnicas de optimización se han empleado para solucionar el problema de asignar los grupos de eventos en los períodos de tiempo [FS83], [BLW92]. La desventaja principal de estos métodos es que los conjuntos de eventos son formados y fijados al principio del algoritmo y éste puede dar lugar a una programación de horarios de costo alto.

2.3.4. Enfoque Basado en Restricciones

En estos métodos el TTP se modela como un conjunto de variables (eventos) a los cuales los valores (recursos tales como salones y períodos de tiempo) son asignadas de manera de satisfacer un número de restricciones [BPS99], [Whi00]. Generalmente un número de reglas son definidas para asignar los recursos a los eventos. Cuando no hay una regla que se pueda aplicar a una solución parcial, se realiza un "backtracking" hasta que se encuentre una solución.

Problema de Satisfacción de Restricciones

Un Problema de Satisfacción de Restricciones (Constraint Satisfaction Problem, CSP) es definido como un conjunto V de variables, un dominio D finito y discreto de valores potenciales para cada variable, y un conjunto C de restricciones que especifican las combinaciones de valores de las variables que sean aceptables. Una asignación de valores a cada una de las variables de manera que todas las restricciones sean satisfechas constituye una solución del problema.

Un método muy usado para solucionar el TTP es tratarlo como un CSP. Los CSP son problemas que han llamado la atención de los investigadores en Inteligencia Artificial, puesto que aparecen en muchas áreas, por ejemplo, visión, razonamiento temporal y asignación de recursos, ver por ejemplo [Tsa93] y [FM94].

Un CSP es definido por Tsang [Tsa93] como: *"A problem composed of a finite set of variables, each of which is associated with a finite domain, and a set of constraints that restricts the values the variables can simultaneously take. The task is to assign a value to each variable satisfying all the constraints"*.

Un clásico ejemplo de CSP es el Problema N-Reinas (N-Queens Problem) que ha sido usado extensivamente para mostrar los resultados de las investigaciones en los algoritmos CSP, ver por ejemplo [MPL90]. En [MEsG93] se aplica CSP al TTP. Duncan [Dun93] describe varias herramientas comerciales para CSP.

El método mas popular para implementar CSP es Constraint Logic Programming que proporciona un lenguaje de programación en el cual se especifican las restricciones.

Programación Lógica de Restricciones

Programación Lógica de Restricciones (Constraint Logic Programming, CLP) se basa en la integración del CSP en un esquema de programación lógica (Prolog). Esta combinación ayuda a hacer los programas de CLP más expresivos y flexibles, y en algunos casos, más eficientes que otra clase de programas.

Tres herramientas bien conocidas que se pueden utilizar para construir horarios son:

- CHIP (Constraint Handling in Prolog). Explicado en [DHS+88], fue originalmente desarrollado en el European Computer Industry Research Centre[d] (ECRC) en Munich, un centro de investigación europeo fundado conjuntamente por Bull[e], Siemens[f] e ICL[g], es un lenguaje de programación lógica de restricciones basado en Prolog que combina técnicas de consistencia con "backtracking". Los resultados fueron explotados por Bull en 1989 cuando rediseñaron el sistema y produjeron una nueva versión llamada CHARME, la cual tiene varias de las cosas de CHIP pero mas orientado a resolver problemas de planificación de actividades. Un buen tutorial de como usar CHARME puede ser encontrado en [DJ91]. CHIP ha sido aplicado al TTP en los siguientes artículos [BGJ94], [GJBP96] y [Laj96]. Otros trabajos relacionados son [KG96], [SVK97], [Gol00] y [AM00].
- ILOG[h]. Fue desarrollado también por el ECRC pero en otra dirección. Se relaciona mas de cerca con el desarrollo de Le-Lisp en el centro de investigación francés INRIA[i]. Tiene una serie de componentes que se dedican a solucionar problemas específicos, especialmente los relacionados con la programación de actividades y la optimización. Proporciona bibliotecas para manejar varias clases de estrategias y restricciones y tiene la capacidad de incorporar otras heurísticas. Los módulos principales son el ILOG Solver, ILOG Scheduler y el ILOG Planner. El ILOG Solver es la base del sistema y proporciona la tecnología para modelar las relaciones y restricciones de los problemas.

Otros lenguajes declarativos que fueron desarrollados para los diferentes TTP, incluyendo WPROLOG, se pueden ver en [KW92], COASTOOL [YMNW94], Oz [HW96] y EaCL [TW99].

En las investigaciones del TTP las técnicas basadas en las restricciones se utilizan para modelar el problema como un CSP. La asignación de variables afecta la eficacia de las diferentes heurísticas de búsqueda. La mayoría de los estudios con CLP dieron buenas soluciones factibles, que fueron mejoradas empleando diversas técnicas. Por ejemplo, CLP ha sido usado para producir el punto de partida para aplicar Búsqueda Tabú, el cual no solo produce mejores resultados sino que también reduce el tiempo computacional [WZ97]. En [YMNW94] se propuso una relajación de las restricciones del problema para producir una buena asignación inicial, que entonces fue mejorada usando hill-climbing para un TTP real.

[d] ECRC - http://www.ecrc.de
[e] Bull - http://www.bull.com
[f] Siemens - http://www.siemens.de
[g] ICL - International Computers Limited - http://www.icl.co.uk
[h] ILOG - http://www.ilog.com
[i] INRIA - Institut National de Recherche en Informatique et en Automatique - http://www.inria.fr

2.3.5. Métodos Metaheurísticos

Las Metaheurísticas [CDM+96] surgen derivadas de la observación de los procesos que ocurren en los fenómenos físicos, biológicos y sociales de la naturaleza, extrayendo de esos fenómenos ciertas características que pueden ser aplicadas en la resolución de los problemas de optimización combinatoria $\mathcal{NP}$-completos. Una de las características que tienen las metaheurísticas es que han sido definidas de manera de que sean robustas (aplicables a una gran variedad de problemas con una mínima, sí fuera necesario, modificación de su definición básica) y efectivas como herramientas de optimización, las cuales comienzan con una solución inicial o una población de individuos iniciales y se van modificando iterativamente hasta que se llegue a la condición de fin, la cual se puede definir como haber obtenido una solución considerada aceptable o bien llegar al número máximo de iteraciones.

Cada metaheurística cumple que:

- Se aplica utilizando cierto número de ciclos, hasta satisfacer criterio de fin.
- Utiliza uno o más "agentes" (neuronas, partículas, cromosomas, hormigas, etc.).
- En caso de múltiples agentes tiene definido un mecanismo de competición-cooperación.
- Permite modificar los parámetros y la representación del problema.

En las últimas dos décadas una variedad de metaheurísticas entre las que se encuentran Simulated Annealing, Búsqueda Tabú, Algoritmos Genéticos y Ant System han sido investigadas para el TTP.

A continuación se presenta una breve descripción de las metaheurísticas Optimización por Colonia de Hormigas, Búsqueda Tabú, Simulated Annealing, Métodos de la Pendiente, Algoritmos Genéticos y Algoritmos Meméticos y su aplicación al TTP. En el apéndice A se presenta la descripción detallada de Algoritmos Genéticos, Búsqueda Tabú y Optimización por Colonia de Hormigas. Y en el capítulo 4 la adaptación de las mismas al problema de programación de horarios de la Facultad.

Optimización por Colonia de Hormigas

Optimización por Colonia de Hormigas (Ant Colony Optimization, ACO) [DC99], [DCG99] es una metaheurística propuesta recientemente para resolver problemas de optimización combinatoria de difícil solución. La fuente de inspiración de la metaheurística es el comportamiento de las hormigas para establecer la ruta mas corta entre el hormiguero y una fuente de alimentos, utilizando feromonas como medio de

comunicación. En analogía con el ejemplo biológico, ACO se basa en la comunicación indirecta de una colonia de agentes, llamadas hormigas (artificiales), medida por los rastros (artificiales) de feromona. Los rastros de feromona en ACO sirven como información distribuida y numérica que las hormigas usan para construir probabilísticamente la solución al problema y que las hormigas adaptan durante la ejecución de los algoritmos para reflejar sus experiencias de búsqueda.

Las hormigas artificiales usadas en ACO construyen probabilísticamente una solución agregando iterativamente componentes a las soluciones parciales considerando (i) la información heurística de la instancia del problema que se está solucionando, si está disponible y (ii) los rastros (artificiales) de feromona que cambian dinámicamente con el tiempo reflejando la experiencia adquirida por los agentes en la búsqueda.

La interpretación de ACO como extensión de las heurísticas constructivas es debido a varias razones. Primero, un componente estocástico en ACO permite que las hormigas construyan una variedad amplia de soluciones y por lo tanto exploran un número mucho mas grande de soluciones que una heurística golosa (greedy). Al mismo tiempo, el uso de la información heurística (ver sección A.4.2) de la instancia del problema que se está resolviendo, que está disponible fácilmente para muchos problemas, puede dirigir las hormigas hacia soluciones más prometedoras. Lo mas importante es que se puede utilizar la experiencia de búsqueda de las hormigas influenciando en la manera que el algoritmo construye la solución en las futuras iteraciones.

El primer algoritmo ACO propuesto fue Ant System (AS). AS fue aplicado a instancias pequeñas del Problema del Viajante de Comercio (Traveling Salesman Problem, TSP) con menos de 75 ciudades. Originalmente AS fue un conjunto de tres algoritmos llamados ant-cycle, ant-density y ant-quantity. Estos tres algoritmos fueron propuestos en la tesis de doctorado de Dorigo [Dor92] y primeramente aparecieron en [DMC91] y [DMC92] y luego en otros trabajos [CDM92] y [DMC96]. Mientras que en ant-density y ant-quantity las hormigas depositan la feromona luego de moverse de una ciudad a otra, en ant-cycle la actualización de la feromona es hecha después de que todas las hormigas construyan su solución y la cantidad depositada por cada hormiga es fijada según la calidad de la solución obtenida. Debido a que ant-cycle presentaba mejor performance que las otras dos variantes, fue mas tarde llamado simplemente Ant System y los otros dos algoritmos dejaron de ser estudiados.

Los resultados computacionales de AS eran prometedores pero no competitivos, por ello, el mérito principal de AS fue el de estimular a los investigadores, sobre todo de Europa, a desarrollar extensiones y mejorar sus ideas básicas con otros métodos [DC99].

Existen al menos dos variantes básicas de ACO, el $\mathcal{MAX} - \mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) propuesto inicialmente en [SH00] y el Ant Colony System (ACS) que es descrito en detalle en [DG97b]. Mientras que la idea básica es idéntica para ambas

variaciones, hay algunas diferencias principalmente en la manera en que la feromona es actualizada.

AS es mejorado por ACS [GD96], [DG97a], [DG97b] aumentando la importancia de la información recogida por las hormigas anteriores con respecto a la exploración del espacio de búsqueda. Esto se alcanza a través de dos mecanismos: primero, una estrategia elitista fuerte se utiliza para actualizar los rastros de feromona permitiendo solo a la hormiga que ha producido la mejor solución actualizar finalmente el rastro y en segundo lugar, las hormigas eligen el siguiente movimiento utilizando una regla pseudoaleatoria (ver sección A.4.4). Finalmente, el algoritmo ACS se diferencia de los algoritmos ACO anteriores en que las hormigas actualizan los rastros de feromona mientras construyen soluciones (como en ant-quantity y ant-density). En la práctica, las hormigas ACS "comen" algo del rastro de la feromona en los caminos que visitan. Esto tiene el efecto de disminuir la probabilidad que una misma trayectoria sea utilizada por todas las hormigas (es decir, favorece la exploración, contrapesando de esta manera a los otros algoritmos que favorecen fuertemente la explotación del conocimiento adquirido del problema).

$\mathcal{MM}$AS [SH97], [Stü99], [SH00] introduce límites superiores e inferiores para los rastros de feromona, así como una manera diferente de inicializar esos valores. En la práctica el rastro de feromona es limitado por un valor mínimo y un valor máximo e inicialmente se le da el valor máximo que causa una exploración mas amplia al comienzo del algoritmo. También, como en ACS, en $\mathcal{MM}$AS solamente la mejor hormiga se le permite agregar feromona después de cada iteración del algoritmo. Los resultados computacionales han demostrado que los mejores resultados son obtenidos cuando se realizan las actualizaciones de los rastros con la mejor solución obtenida en cada iteración. Similar a ACS, $\mathcal{MM}$AS explota a menudo una búsqueda local para mejorar su funcionamiento.

Ambos algoritmos se han probado para casos del UCTTP obteniendo resultados significativos. ACS fue comparado recientemente con otras metaheurísticas en [RDSC$^+$02], y $\mathcal{MM}$AS fue comparado con Random Restart Local Search en [SKS02] obteniendo mejores resultados.

Según Dorigo y Stützle en el libro "Ant Colony Optimization" [DS04] (pág. 167) el único UCTTP tratado por algún algoritmo ACO es el que se enmarca dentro de las actividades de investigación del proyecto Europeo "Metaheuristic Network"[j] presentado en [RDSC$^+$02] y [SKS02].

En la Facultad un proyecto de grado [RC99] utilizó AS para resolver el caso particular de un UCTTP.

[j]Metaheuristic Network - http://www.metaheuristics.org

Búsqueda Tabú

Búsqueda Tabú (Tabu Search, TS) es una técnica de búsqueda local diseñada para resolver problemas de optimización. Los orígenes de TS puede situarse en diversos trabajos publicados a finales de la década del 70 [Glo77]. Oficialmente, el nombre y la metodología fueron introducidos posteriormente por Fred Glover en 1989 y 1990 [Glo89], [Glo90] y específicamente aplicado al QAP por Skorin-Kapov [SK90] y Taillard [Tai90]. Numerosas aplicaciones han aparecido en la literatura, así como artículos y libros para difundir el conocimiento teórico del procedimiento [GL97].

TS es un procedimiento heurístico utilizado para resolver problemas de optimización de gran escala, el cual está diseñado para guiar a otros procedimientos de búsqueda local (procesos componentes) para escapar de la optimalidad local y poder explorar mejor y más extensamente el espacio de soluciones. Tiene como antecedentes a varios métodos diseñados para cruzar fronteras de factibilidad o de optimalidad local; sistemáticamente impone y relaja restricciones para permitir la exploración de regiones de otra manera prohibidas. TS utiliza la estrategia de evitar que la búsqueda quede atrapada en un óptimo local que no sea global, para eso utiliza el concepto de memoria tratando de extraer información de lo ya sucedido y actuar en consecuencia.

La técnica TS está basada en la noción de vecindad, dado un problema de optimización P, sea S el espacio de soluciones factibles del problema P y la función de costo f definida en S. Una función $\mathcal{N}$, que depende de la estructura del problema, que asigna a cada solución factible $s \in S$ sus vecinos $\mathcal{N}(s) \subseteq S$ y cada solución $s' \in \mathcal{N}(s)$ es llamada vecina de s.

TS comienza con una solución inicial s, que puede ser obtenida con otra técnica o generada en forma aleatoria a la cual se le calcula el costo $f(s)$ y se explora un subconjunto V de los vecinos $\mathcal{N}(s)$, de estas se elige la mejor, independientemente si ésta mejora o no el valor de la función objetivo en s. La característica principal es precisamente la construcción de una lista tabú de movimientos, aquellos movimientos que no son permitidos en la presente búsqueda. La razón de la lista tabú es la de excluir la posibilidad de regresar a puntos visitados en alguna búsqueda anterior. Pero un movimiento permanece en la lista por un cierto número de búsquedas, de forma que se tiene una lista cíclica en donde la nueva solución es agregada y se elimina la más vieja.

La lista tabú tiene la meta de prevenir ciclos e inducir la exploración de nuevas regiones. Pero también una buena trayectoria de búsqueda resultará al re-visitar una solución encontrada anteriormente. Ahora bien, las restricciones tabú no son inviolables para toda circunstancia. Cuando un movimiento tabú proporciona una solución mejor que cualquier otra previamente encontrada, su clasificación como tabú puede eliminarse. La condición que permite dicha eliminación se llama Criterio de Aspiración.

Es así como las restricciones tabú y el criterio de aspiración juegan un papel complementario en la restricción y guía del proceso de búsqueda. Las restricciones tabú permiten que un movimiento sea admisible si no está clasificado como tabú, mientras que si el criterio de aspiración se satisface, permite que un movimiento sea admisible aunque esté clasificado como tabú. En forma simple TS descubre dos de sus elementos claves: la de restringir la búsqueda mediante la clasificación de ciertos movimientos como prohibidos (tabú) y el de liberar la búsqueda mediante una función de memoria de término corto que proporciona una estrategia de olvido.

Hertz y de Werra aplicaron TS al GCP [HdW87]. Mas tarde al TTP en [Her91], [Cos94] y [Sch96].

En [Her91] las restricciones son agrupadas en rígidas (duras, hard) y flexibles (blandas, soft). Sin embargo, una solución factible no es definida basándose en las restricciones rígidas porque esto no garantiza que el correspondiente espacio de búsqueda esté conectado (relación de vecindad). Por esta razón, el concepto de solución factible para el proceso de búsqueda es diferente a la factibilidad del TTP. De modo que el proceso puede pasar a través de soluciones que no cumplan las restricciones. En particular, Hertz considera que una solución es factible aunque tenga cursos programados a la misma hora con profesores o alumnos en común. Todas las otras restricciones rígidas son satisfechas por las soluciones factibles. La vecindad $\mathcal{N}(s)$ de una solución s consiste en todos los horarios que se pueden obtener de s asignando una clase l a otro período diferente t. Notar que basado en la definición anterior de solución factible, esta operación siempre resulta en una solución factible.

Nonobe e Ibaraki [NI98] desarrollaron una herramienta basada en TS para varios tipos de CSP incluido el TTP. Alvarez-Valdes, Crespo y Tamarit [AVCT02] desarrollaron un sistema con interfaz de usuario basado en TS con un conjunto de heurísticas. La integración de TS con otras técnicas también ha sido investigado, por ejemplo en [WZ97].

Descripción detallada de TS y desarrollos recientes pueden ser encontrados en [GLT93], [GL93], [Glo97], [GL02], [Gen02] y [Gen03].

Simulated Annealing

Kirkpatrick, Gelatt y Vecchi proponen en 1983 [KGV83] un método para obtener soluciones aproximadas a problemas de optimización, llamado Simulated Annealing (SA). Este procedimiento se basa en una analogía con el comportamiento de un sistema físico al someterlo a un baño de agua caliente. SA ha sido probado con éxito en numerosos problemas de optimización, mostrando gran "habilidad" para evitar quedar atrapado en óptimos locales. Debido a su sencillez de implementación así como a los buenos resultados que fueron apareciendo, experimentó un gran auge en la década

de los 80.

Kirpatrick y otros, trabajando en el diseño de circuitos electrónicos, consideraron aplicar el algoritmo de Metrópolis [MRTT53] en alguno de los problemas de optimización combinatoria que aparecen en este tipo de diseños. El algoritmo de Metrópolis simula el cambio de energía en el proceso de enfriamiento de un sistema físico. Las leyes de la termodinámica establecen que a una temperatura t la probabilidad de un aumento de energía de magnitud ∂E viene dada por la expresión siguiente:

$$P(\partial E) = e^{\frac{-\partial E}{kt}}$$

donde k es la constante de Boltzmann.

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

Kirpatrick y otros, pensaron que se podía establecer una analogía entre los parámetros que intervienen en la simulación termodinámica de Metrópolis y los que aparecen en los métodos de optimización local. Para ello, establecen un paralelismo entre el proceso de las moléculas de una sustancia que van colocándose en los diferentes niveles energéticos buscando un equilibrio, y las soluciones visitadas por un procedimiento de búsqueda local. Así pues, SA es un procedimiento basado en búsqueda local en donde todo movimiento de mejora es aceptado y se permiten movimientos de no mejora de acuerdo con ciertas probabilidades. Dichas probabilidades están basadas en la analogía con el proceso físico de enfriamiento y se obtienen como función de la temperatura del sistema.

La estrategia de SA es comenzar con una temperatura inicial alta, lo cual proporciona una probabilidad también alta de aceptar un movimiento de no mejora. En cada iteración se va reduciendo la temperatura y por lo tanto las probabilidades son cada vez más pequeñas conforme avanza el procedimiento y nos acercamos a la solución óptima. De este modo, inicialmente se realiza una diversificación de la búsqueda sin controlar demasiado el costo de las soluciones visitadas. En iteraciones posteriores resulta cada vez más difícil el aceptar "malos" movimientos y por lo tanto se produce un descenso en el costo de la función objetivo. De esta forma, SA tiene la habilidad de "salir" de óptimos locales al aceptar movimientos de no mejora en los estados intermedios. Al final del proceso éstos son tan poco probables que no se producen, con lo que, si no hay movimientos de mejora, el algoritmo finaliza.

Los estudios de Kirpatrick y otros fueron extendidos en [vLA87] y [AK89b].

La técnica comienza creando una solución aleatoria inicial y el procedimiento principal genera en forma aleatoria en cada iteración un vecino de la solución actual (similar a

TS). Sea Δ la diferencia en la función objetivo entre la nueva solución y la actual. Si $\Delta < 0$ la nueva solución es aceptada (minimizando la función objetivo) y si $\Delta \geq 0$ la nueva solución es aceptada con probabilidad $e^{\frac{\Delta}{T}}$, donde T es un parámetro llamado temperatura.

Abramson [Abr91] presentó un estudio de SA y que fue puesto en ejecución en un multiprocesador y presentó futuras investigaciones. Algunos trabajos concluyen que la implementación de SA en el problema de los TTP de exámenes [Bul98], [TD98] y en el de cursos [ECF98], [MCR98], [ADK99] es fuertemente dependiente de los ajustes y los parámetros (espacio de soluciones, enfriamiento, generación de los vecinos, la función de costo).

Métodos de la Pendiente

Existen pocos estudios que aplican los Métodos de la Pendiente (Descent Methods, DM) como solución al TTP, ver [RC95].

Los DM son mas simples que TS y SA en el sentido de que no tienen ningún mecanismo para escapar del óptimo local. El proceso trabaja eligiendo un movimiento vecino descendente y si no existe tal movimiento el algoritmo termina en el óptimo local. Un movimiento vecino descendente es elegido de dos maneras: el movimiento de mayor pendiente o seleccionado por azar. La búsqueda del movimiento de mayor pendiente permite al método recorrer la vecindad entera o al menos un conjunto de ella, se calcula y se elige el movimiento que conduce al aumento más grande de la calidad de la solución. Este proceso es determinista y alcanza rápidamente el óptimo local. El movimiento seleccionado por azar da la oportunidad de explorar las áreas planas del espacio de soluciones. La naturaleza aleatoria del proceso presenta la oportunidad de volver a una solución anterior y de intentar quizás una ruta de búsqueda diferente.

Algoritmos Genéticos

Los Algoritmos Genéticos (Genetic Algorithms, GA) fueron introducidos primero por Holland [Hol75] como un algoritmo robusto de búsqueda. Mas tarde, especialmente en los trabajos de Goldberg [Gol89] fueron utilizados como técnica de optimización.

Los GA son métodos adaptativos que están basados en los procesos genéticos de organismos biológicos, codificando una posible solución al problema en un cromosoma compuesto por una cadena de bits o caracteres. Estos cromosomas representan individuos que son llevados a lo largo de varias generaciones, en forma similar a las poblaciones naturales, evolucionando de acuerdo a los principios de selección natural y supervivencia del más apto, descritos por primera vez por Charles Darwin en su libro “Origen de las Especies”. Emulando estos procesos, los GA son capaces de hacer

evolucionar soluciones de problemas del mundo real. En la naturaleza, los individuos compiten entre sí por recursos tales como comida, agua y refugio. Adicionalmente, los animales de la misma especie normalmente antagonizan para obtener una pareja. Aquellos individuos que tengan más éxito tendrán probablemente mayor cantidad de descendientes, por lo tanto, mayores probabilidades de que sus genes (unidad básica de codificación de cada uno de los atributos de un ser vivo) sean propagados a lo largo de sucesivas generaciones. La combinación de características de los padres mejor adaptados, en un descendiente, puede producir muchas veces un nuevo individuo mejor adaptado que cualquiera de sus padres a las características del medio ambiente. Holland a fines de la década del 60 estaba consciente de la importancia de la selección natural y desarrolló una técnica que permitió incorporarla en un programa de computadora. Su objetivo era lograr que las computadoras aprendieran por sí mismas. A la técnica que inventó Holland se le llamó originalmente Planes Reproductivos, pero se hizo popular bajo el nombre de Algoritmo Genético tras la publicación de su libro [Hol75]. Una definición completa dada por John Koza [Koz92] dice que es: *"Un algoritmo matemático altamente paralelo que transforma un conjunto de objetos matemáticos individuales con respecto al tiempo usando operaciones modeladas de acuerdo al principio Darwiniano de reproducción y supervivencia del más apto y tras haberse presentado de forma natural una serie de operaciones genéticas de entre las que se destaca la recombinación sexual. Cada uno de estos objetos matemáticos suele ser una cadena de caracteres (generalmente letras o números) de longitud fija que se ajusta al modelo de las cadenas de cromosomas y se les asocia con una cierta función matemática que refleje su aptitud."*

Los GA utilizan una analogía directa del fenómeno de la evolución en la naturaleza. Trabajan con una población de individuos y cada uno representando una posible solución a un problema dado. A los individuos más adaptados se les da la oportunidad de reproducirse mediante cruzamientos con otros individuos de la población produciendo descendientes con características de ambos padres. Los miembros menos adaptados poseen pocas probabilidades de que sean seleccionados para la reproducción y desaparecen. Una nueva población de posibles soluciones es generada mediante la selección de los mejores individuos de la generación actual, emparejándolos entre ellos para producir un nuevo conjunto de individuos. Esta nueva generación contiene una proporción más alta de las características poseídas por los mejores miembros de la generación anterior. De esta forma, a lo largo de las diferentes generaciones, las características buenas son difundidas a lo largo de la población mezclándose con otras. Favoreciendo el emparejamiento de los individuos mejor adaptados es posible recorrer las áreas más prometedoras del espacio de búsqueda.

Existen tres operaciones básicas que contribuyen a la evolución:

- Reproducción: cada individuo se reproduce según cierta probabilidad.
- Cruzamiento: dos individuos cruzan su material genético, o sea cada uno es

dividido y las partes son cruzadas. De esta manera dos nuevos individuos son creados.

- Mutación: es la introducción en forma aleatoria de un nuevo individuo, usualmente producto del cambio de un gen de un individuo. Esta operación no debe ser utilizada demasiado (generalmente en el orden de 1 en 1000) ya que el GA se puede convertir en una búsqueda al azar, pero su utilización asegura que ningún punto en el espacio de búsqueda tenga probabilidad 0 de ser examinado.

Dado un problema de optimización, se define la Función de Aptitud (Fitness) como la función objetivo del problema y cada individuo de la población representa una de las soluciones factibles. Los GA tienen como objetivo encontrar un individuo de la población que tenga el mejor valor de aptitud. Inicialmente el algoritmo define en el tiempo 0 una población inicial $S_1^0, S_2^0, ..., S_p^0$ de p individuos, generalmente seleccionados de forma aleatoria, para la cual se busca cual es el mejor individuo, donde cada individuo consiste en una colección (usualmente un string) de elementos atómicos llamados "genes". Cada gen toma su valor en un conjunto predeterminado. En cada paso t del algoritmo se realiza la generación $S_1^t, S_2^t, ..., S_p^t$ y se trabaja sobre la población modificando sus componentes. Las modificaciones ocurren de acuerdo a las reglas genéticas, implementadas por las operaciones genéticas definidas anteriormente. La reproducción selecciona uno por uno los individuos en $t+1$ a partir de los de t a través de un sorteo. El sorteo se realiza según las probabilidades de reproducción basadas en su función de aptitud. Luego a cada par de individuos de la población obtenida se le aplica el cruzamiento con una probabilidad de cruzamiento p_c. Y finalmente a la población obtenida se le aplica la mutación con probabilidad de mutación p_m. Cuando el criterio de parada es satisfecho se devuelve la mejor solución obtenida. Generalmente se usan dos criterios de parada: correr el algoritmo durante un número máximo de generaciones o detenerlo cuando la población se haya estabilizado, o sea cuando todos o la mayoría tengan la misma aptitud. Los principales parámetros que controlan el método son el tamaño de la población p, la probabilidad de cruzamiento p_c y la probabilidad de mutación p_m.

El poder de los GA proviene del hecho de que la técnica es robusta y puede manejar exitosamente un amplio rango de problemas, incluso algunos que son difíciles de resolver por otros métodos. Los GA no garantizan que encontrarán la solución óptima al problema, pero son generalmente buenos encontrando en corto tiempo buenas soluciones a los problemas. Donde existan técnicas especializadas para la resolución de problemas, éstas superarán fácilmente a los GA tanto en velocidad como en precisión. El campo principal de aplicación es en donde no existan este tipo de técnicas o donde no se puedan aplicar.

En [CDM90] se construye un horario para una escuela secundaria de Italia usando GA y fueron los primeros en reportar el éxito de aplicar GA a los TTP. Davis [Dav91] presenta un algoritmo híbrido GA/greedy para resolver el GCP, que como ya se

vio está relacionado con el TTP. En [Lin92] usan un GA híbrido para asignar los horarios de los profesores en Singapur, utilizan un programa PROLOG inicialmente para resolver la mayor parte del problema y luego GA para resolver el resto del problema. Mientras generalmente GA comienza con una población inicial creada en forma aleatoria, en [EK96] se utiliza un procedimiento basado en PROLOG para obtener poblaciones iniciales de mejor calidad y factibles.

Algoritmos Meméticos

Los Algoritmos Meméticos (Memetic Algorithms, MA) intentan mejorar el comportamiento de los GA incorporando búsqueda local [MN92]. La principal idea de los MA es de explorar la vecindad de la solución obtenida por GA en cada paso buscando y mejorando el óptimo local (para cada solución) antes de pasar el control al GA nuevamente y continuar con el proceso. La idea fue formalizada en [RS94] y comparada con GA.

Un "meme" se puede pensar en una unidad de información que se reproduce a si misma. Un "meme" se diferencia de un "gen" en la manera que es pasado entre los individuos, cada individuo adapta el mejor posible mientras que los genes se pasan inalterados.

La principal ventaja a favor del uso de los MA es que el espacio de soluciones posibles es reducido al subespacio del óptimo local.

En [PCL95], [PCNL96], [Ran96] y [PRC98] se utilizaron los MA para el UCTTP y en [BNW96] para el UETTP.

2.3.6. Métodos Hiperheurísticos

Las Hiperheurísticas se pueden definir como heurísticas que eligen heurísticas [CKS00], [CKS01] y [CKH02]. La diferencia principal entre las hiperheurísticas y las metaheurísticas extensamente usadas en los TTP es que la primera es un método que usa una heurística que selecciona otras heurísticas que pueden incluir a las metaheurísticas. De modo que la hiperheurística es potencialmente un método de uso general.

En [TRV99] usan GA para seleccionar la heurística que ordena los exámenes en un TTP utilizando satisfacción de restricciones. Cowling, Kendall y Sobiega [CKS00] [CKS01] utilizaron una hiperheurística para seleccionar de un conjunto de heurísticas de nivel inferior según las características del espacio de búsqueda en un SP.

2.3.7. Otros métodos

Redes Neuronales

La aplicación de Redes Neuronales (Neural Nets, NN) a los TTP siempre pareció prometedora, pero no hay muchos resultados a señalar en este tiempo. Una aproximación común es utilizar NN para encontrar soluciones a los problemas de optimización combinatoria, por ejemplo Roos y Hallam [RJ93] discuten cómo aplicar una red de Hopfield para solucionar el Problema del Viajante de Comercio (Traveling Salesman Problem, TSP). Diversos investigadores han utilizado diversos modelos de la redes neuronales para TTP y SP. Por ejemplo, Gislen, Peterson y Soderberg [GPS89] han utilizado para un TTP pequeño NN de Potts y en 1992 [GPS92] lo ampliaron a TTP más grandes. Varios modelos de NN se han aplicado con éxito al TSP, ver [HT85], [DW87], [ACT88], [AK89a], [FW91] y [YK92].

Sistemas Expertos

Una técnica basada en Sistemas Expertos (Expert Systems, ES) es dada en [MGK91], [SGM94]. En [SGM94] se define un lenguaje basado en reglas, llamado RAPS, para especificar el problema general de ubicación de recursos y el uso, entre otros, para un UCTTP. En particular ellos consideran las clases como actividades y los períodos de tiempo como recursos a ser asignados a las actividades.

Otros autores [Mon88], [PCSD89] y [DR90] hacen uso de los ES. Por ejemplo en [DR90] proponen el uso de un sistema experto llamado PROTEUS, para la asignación de profesores a los cursos y lo comparan con las técnicas de programación entera.

Sistemas Basados en el Conocimiento

Una técnica raramente usada en el TTP es el uso de sistemas basados en reglas que simulan las acciones que realiza una persona experimentada en programar los horarios. Johnson [Joh93] comenta que el juicio humano será siempre dominante en la creación de los horarios factibles y aceptables para los cursos. Sugiere que la ayuda más importante que la computadora puede dar es en la forma de almacenaje de información. Con el uso de sistemas basados en conocimiento se podría quizás simular el juicio humano suficientemente para reducir al mínimo la interacción necesaria del usuario.

Pocas publicaciones se ocupan del uso de sistemas basados en el conocimiento aplicados al TTP. En [GKM90] los autores presentan un paradigma general para solucionar problemas de asignación de recursos y de programación de horarios. Los autores tam-

bién realizaron experimentos con tres problemas de la vida real. Otros trabajos en [MGS96] y [RS96].

Capítulo 3

Problema de Asignación de Salones y Horarios a Asignaturas

3.1. Descripción del Problema

El Problema de Asignación de Salones y Horarios a Asignaturas (PASHA) es un UCTTP orientado a la organización de las carreras de la Facultad de Ingeniería de la Universidad de la República [RC99], [CPU03]. El problema busca resolver la asignación de las asignaturas de todas las carreras de la Facultad para un semestre dado. Esta asignación debe cumplir con ciertas restricciones que surgen de tener en cuenta la organización de la Facultad, los recursos disponibles en cada salón y la cantidad de alumnos inscriptos en las diferentes asignaturas. Es deseable además, que se satisfagan ciertas preferencias que, si bien no son obligatorias, contribuyen a que la solución encontrada sea de mejor calidad.

3.1.1. Organización de la Facultad

La actividad académica en la Facultad se organiza en Carreras, Años, Semestres, Asignaturas, Grupos y Clases.

Carreras, Años y Semestres

Cada carrera, por ejemplo: Ingeniería Civil, Ingeniería Mecánica, Ingeniería en Computación, etc., tiene una duración de varios años. A su vez cada año se divide en dos semestres.

Asignaturas

Una asignatura o curso puede pertenecer a más de una carrera y no necesariamente al mismo año dentro de ellas, pudiendo además tener una identificación diferente dependiendo de la carrera. Esas asignaturas comunes a las carreras son agrupadas en conjuntos de asignaturas equivalentes y se identifica una asignatura primaria, que es la única que se considera en representación de las otras para la asignación.

Las asignaturas tienen diferentes niveles de importancia, los cuales determinan las prioridades a la hora de establecer la asignación. Por ejemplo, las asignaturas del ciclo básico tienen mayor prioridad que las de las otras carreras y dentro de una misma carrera son más importantes las asignaturas obligatorias que las electivas. Por lo tanto se definen los siguientes niveles en orden de importancia:

- Básicas: Asignaturas pertenecientes al ciclo básico de la Facultad de Ingeniería, deben tener precedencia al momento de asignarlas.
- Obligatorias: Asignaturas que son obligatorias para las carreras a las cuales pertenece.
- Electivas: Asignaturas electivas para alguna o todas las carreras a las cuales pertenece, su importancia dentro de la asignación es menor a la de las obligatorias.

Debido a que las asignaturas pueden pertenecer a más de una carrera, el criterio de importancia es una característica de la relación entre una asignatura y una carrera, ya que una misma asignatura puede ser obligatoria para una carrera y electiva para otra.

Generalmente, se tienen estimados para cada asignatura los porcentajes de abandono y recursantes dentro de los alumnos inscriptos. Es importante tener en cuenta esos porcentajes al realizar la asignación para determinar si la capacidad de los salones es suficiente para la cantidad de alumnos que efectivamente concurrirán a esa asignatura.

Grupos

Las asignaturas se organizan en uno o más grupos, entre los cuales se distribuyen los alumnos inscriptos a las mismas. Los grupos pueden ser teóricos o prácticos y cada asignatura tiene al menos un grupo teórico.

Clases

Cada grupo de cada asignatura es dividido en clases. Una clase es la unidad básica a asignar a los salones y horarios. Cada clase tiene una duración, días y horarios de preferencia. La cantidad de alumnos que asisten a una clase queda determinada por la cantidad de alumnos asignados al grupo al cual pertenece.

Existen asignaciones de clase que deben ser fijas, es decir que el proceso de búsqueda de una asignación no puede modificar (por ejemplo las clases dictadas en algunos Institutos, que ya vienen prefijadas en su horario y salón).

Una clase puede necesitar que el salón en que se dicta disponga de cierta infraestructura, como por ejemplo enchufes, pantallas y/o cortinas.

Turnos

Debido a la gran cantidad de estudiantes que ingresan en primer año se particiona el dictado de las clases en turnos. Para cada clase se determina cual es el turno que le corresponde. Se definen los siguientes turnos:

- Mañana
- Tarde
- Noche
- Todo el día, para las asignaturas que no tengan preferencias horarias.

Salones

La principal característica que se tiene en cuenta de los salones es su capacidad locativa. También se tiene en cuenta las características de infraestructura (recursos) y datos informativos como el nombre y la ubicación.

Existen asignaciones de clases a horarios que son fijas, por lo cual no deben modificarse en el proceso de asignación. Del mismo modo, existen salones que no se pueden utilizar libremente, y no se pueden tomar en cuenta en la asignación automática.

Se definen tres estados posibles para los salones:

- Asignación Libre: Salón en uso, disponible para el proceso automático de asignación.

- Asignación Limitada: Salón en uso, solo utilizable en asignaciones manuales, por ejemplo los salones pertenecientes a los distintos Institutos, los cuales son asignados por los mismos Institutos.
- No Disponible: Salón en desuso, no disponible para el proceso automático de asignación, ni para asignaciones manuales.

Recursos

Existen clases que requieren de cierta infraestructura para que su dictado sea posible, que se llaman recursos. Por ejemplo proyectores, pizarrones especiales, cortinas, etc.

Horarios de dictado de Clases

Las clases se dictan desde las 8:00 a las 23:00 horas de lunes a viernes y los sábados de 8:00 a 12:00 horas y la unidad mínima de dictado es media hora. Por lo tanto la duración de cada clase se expresa en múltiplos de la misma.

3.1.2. Restricciones y Preferencias

Una asignación del PASHA debe cumplir con todas las restricciones rígidas llamadas Restricciones y satisfacer de la mejor manera posible las restricciones flexibles llamadas Preferencias.

Restricciones

1. Cada clase tiene una duración determinada.
2. Una clase tiene un único salón.
3. Un horario y salón se programan a una única clase.
4. Las clases se dictan dentro de los horarios válidos.
5. Todas las clases son programadas.
6. El horario de cada clase es contiguo.
7. Cada clase se dicta en el mismo día.
8. Las clases de un grupo deben estar en el turno especificado.
9. Las clases de un grupo deben dictarse en diferentes días.

Preferencias

1. Superposición: Las clases de una asignatura básica u obligatoria con un solo grupo no se superponen.
2. Preferencia de Horario: Todas las clases se programan dentro del horario de preferencia especificado.
3. Capacidad del Salón: La capacidad del salón es suficiente para la clase.
4. Recursos Necesarios: El salón de una clase cuenta con los recursos necesarios para el dictado de la misma.
5. Contigüidad Horaria: Los teóricos y prácticos de una asignatura se dictan en horarios contiguos.
6. Desperdicio de Salón: No se utilizan salones de gran capacidad para clases con pocos alumnos.
7. Espaciamiento en Días: Las clases de un grupo teórico se distribuyen en forma uniforme en la semana. Para evaluar el espaciamiento entre las clases se mide la distancia en días entre la primera y la última en la semana. Dependiendo de la cantidad de clases por semana existe un mínimo exigible para esa distancia:
   - 1 clase por semana: no aplicable
   - 2 clases por semana: distancia ≥ 2 (o sea al menos un día libre entre ambas clases).
   - 3 clases por semana: distancia ≥ 3.
   - 4 clases por semana: distancia ≥ 4.
   - 5 clases por semana: distancia ≥ 5.
   - 6 clases por semana: no aplicable.
8. Similitud de Horario: Las clases de un grupo teórico comienzan en "horario similar" durante la semana. Se considera "horario similar" para las clases de un grupo teórico si el horario de cada una de ellas coincide en por lo menos media hora. No se aplica si la Preferencia de Horario para cada una de esas clases es disjunta.

3.2. Formalización del PASHA

Para la formalización del PASHA se definen los siguientes conjuntos:

- C: Clases

- S: Salones
- H: Horarios, de toda una semana con intervalos de media hora
- G: Grupos, $G = \{g_i / g_i \subseteq Pot(C), i = 1, ..., n\}$
- A: Asignaturas, $A = \{a_i / a_i \subseteq Pot(G), i = 1, ..., m\}$
- T: Turnos, $T = \{ti / ti \subseteq Pot(H), i = 1, ..., s\}$
- R: Recursos

Se define las siguientes variables de decisión:

$$X : C \times S \times H \longrightarrow \{0, 1\}$$

donde

$$X_{csh} = \begin{cases} 1 & \text{si la clase } c \text{ es programada en el salón } s \text{ y la hora } h, c \in C, s \in S, h \in H \\ 0 & \text{sino.} \end{cases}$$

Se definen las siguientes funciones:

- $duracion : C \longrightarrow N^+$, $duracion(c)$ indica la duración de la clase c en cantidad de 1/2 horas.
- $dia : H \longrightarrow \{1, ..., 6\}$, $dia(h)$ indica el día de la semana del horario h.
- $hini : C \longrightarrow H$, $hini(c) = \text{mín}_{h \in H}\{h / X_{csh} = 1, \forall s \in S\}$ indica el horario de inicio de la clase c.
- $hfin : C \longrightarrow H$, $hfin(c) = \text{máx}_{h \in H}\{h / X_{csh} = 1, \forall s \in S\}$ indica el horario de fin de la clase c.
- $turno : C \longrightarrow T$, $turno(c)$ indica el conjunto de horarios del turno válido para la clase c.
- $rango : C \longrightarrow T$, $rango(c)$ indica el conjunto de horarios del rango válido para la clase c.
- $grupo : C \longrightarrow G$, $grupo(c)$ indica el grupo de la clase c.
- $recursosc : C \longrightarrow Pot(R)$, $recursosc(c)$ indica los recursos necesarios para la clase c.

- $recursoss : S \longrightarrow Pot(R)$, $recursoss(s)$ indica los recursos disponibles en el salón s.
- $cpri : G \longrightarrow C$, $cpri(g)$ indica la primer clase de la semana del grupo g.
- $cult : G \longrightarrow C$, $cult(g)$ indica la última clase de la semana del grupo g.
- $hmin : T \longrightarrow H$, $hmin(t)$ indica el horario de inicio del turno t.
- $hmax : T \longrightarrow H$, $hmax(t)$ indica el horario de fin del turno t.
- $dmin : G \longrightarrow N^+$, $dmin(g)$ indica la distancia mínima exigible entre la primera y la última clase de la semana del grupo g.
- $hora : H \longrightarrow N^+$, $hora(h)$ indica la hora "reloj"del horario h.
- $asignatura : G \longrightarrow A$, $asignatura(g)$ indica la asignatura del grupo g.
- $tsignatura : A \longrightarrow \{basica, obligatoria, electiva\}$, $tasignatura(a)$ indica el tipo de la asignatura a.
- $cgrupo : A \longrightarrow N^+$, $cgrupo(a)$ indica la cantidad de grupos de la asignatura a.
- $tgrupo : G \longrightarrow \{teorico, practico\}$, $tgrupo(g)$ indica el tipo del grupo g.
- $capacidad : S \longrightarrow N^+$, $capacidad(s)$ indica la capacidad del salón s.
- $cantalumnos : C \longrightarrow N^+$, $cantalumnos(c)$ indica la cantidad de alumnos de la clase c.

y se definen las siguientes variables auxiliares:

- $Y^{(1)} : C \times S \longrightarrow \{0, 1\}$,
  $Y^{(1)}_{cs} = \begin{cases} 1 & \text{si } \sum_{h \in H} X_{csh} \neq 0 \\ 0 & \text{sino.} \end{cases}$ donde $Y^{(1)}_{cs}$ indica si la clase c es programada en el salón s.
- $Y^{(2)} : C \times C \longrightarrow \{0, 1\}$,
  $Y^{(2)}_{c_1 c_2} = \begin{cases} 1 & \text{si } dia(hini(c_1)) = dia(hini(c_2)) \\ 0 & \text{sino.} \end{cases}$ donde $Y^{(2)}_{c_1 c_2}$ indica si el día de programada la clase c_1 es igual al de la clase c_2.
- $Y^{(3)} : C \longrightarrow \{0, 1\}$,
  $Y^{(3)}_{c} = \begin{cases} 1 & \text{si } \sum_{s \in S} \sum_{\substack{h \in H, \\ h \notin rango(c)}} X_{csh} \neq 0 \\ 0 & \text{sino.} \end{cases}$ donde $Y^{(3)}_{c}$ indica si la clase c es programada fuera del rango de preferencia de horario.

- $Y^{(4)} : C \longrightarrow \{0,1\}$,
  $Y_c^{(4)} = \begin{cases} 1 & \text{si } \sum_{\substack{s \in S, \\ recursosc(c) \not\subseteq recursoss(s)}} \sum_{h \in H} X_{csh} \neq 0 \\ 0 & \text{sino.} \end{cases}$ donde $Y_c^{(4)}$ indica si la clase c es programada en un salón que no tenga los recursos requeridos para la misma.

- $Y^{(5)} : G \longrightarrow \{0,1\}$,
  $Y_g^{(5)} = \begin{cases} 1 & \text{si } (dia(hini(cult(g))) - dia(hini(cpri(g))) < dmin(g)) \\ & \wedge (dia(hmin(rango(cult(g)))) - dia(hmax(rango(cpri(g)))) < dmin(g)) \\ 0 & \text{sino.} \end{cases}$
  donde $Y_g^{(5)}$ indica si las clases de un grupo g no son repartidas en forma uniforme a lo largo de la semana.

- $Y^{(6)} : C \times C \longrightarrow \{0,1\}$,
  $Y_{c_1c_2}^{(6)} = \begin{cases} 1 & \text{si } (hora(hfin(c_1)) \leq hora(hini(c_2)) \vee hora(hfin(c_2)) \leq hora(hini(c_1))) \\ & \wedge (rango(c_1) \cap rango(c_2) \neq \emptyset \\ 0 & \text{sino.} \end{cases}$
  donde $Y_{c_1c_2}^{(6)}$ indica si las clase c_1 y c_2 no comienzan en horas similares a lo largo de la semana.

3.2.1. Restricciones

1. Cada clase tiene una duración determinada.

$$\sum_{s \in S} \sum_{h \in H} X_{csh} = duracion(c) \qquad \forall c \in C$$

2. Una clase tiene un único salón.

$$\sum_{s \in S} Y_{cs}^{(1)} \leq 1 \qquad \forall c \in C$$

3. Un horario y salón se programan a una única clase.

$$\sum_{c \in C} X_{csh} \leq 1 \qquad \forall s \in S, \forall h \in H$$

4. Las clases se dictan dentro de los horarios válidos. Es implícita.

5. Todas las clases son programadas[a].

$$\sum_{s \in S} \sum_{h \in H} X_{csh} \neq 0 \qquad \forall c \in C$$

6. El horario de cada clase es contiguo.

$$hini(c) + duracion(c) - 1 = hfin(c) \qquad \forall c \in C$$

[a] En [CPU03] esta restricción es considerada como preferencia.

7. Cada clase se dicta en el mismo día.

$$dia(hini(c)) = dia(hfin(c)) \qquad \forall c \in C$$

8. Las clases de un grupo deben estar en el turno especificado.

$$\sum_{\substack{h \in H, \\ h \notin turno(c)}} X_{csh} = 0 \qquad \forall c \in C, \forall s \in S$$

9. Las clases de un grupo deben dictarse en diferentes días.

$$\sum_{\substack{c_1 \in C, \\ grupo(c_1)=g}} \sum_{\substack{c_2 \in C, \\ c_1 \neq c_2, \\ grupo(c_2)=g}} Y^{(2)}_{c_1 c_2} = 0 \qquad \forall g \in G$$

3.2.2. Preferencias

1. Superposición[b].

$$costo_1 = \sum_{\substack{c_1 \in C, \\ asignatura(grupo(c_1))=a}} \sum_{\substack{c_2 \in C, \\ c_1 \neq c_2, \\ asignatura(grupo(c_2))=a}} (\sum_{s \in S} X_{c_1 sh} \times \sum_{s \in S} X_{c_2 sh})$$

$$\forall a \in A / tasignatura(a) \neq electiva \wedge cgrupo(a) = 1, \forall h \in H$$

2. Preferencia de Horario.

$$costo_2 = \sum_{c \in C} Y^{(3)}_c$$

3. Capacidad del Salón.

$$costo_3 = \sum_{c \in C} \sum_{\substack{s \in S, \\ capacidad(s) < cantalumnos(c)}} Y^{(1)}_{cs} \times (cantalumnos(c) - capacidad(s))$$

4. Recursos Necesarios.

$$costo_4 = \sum_{c \in C} Y^{(4)}_c$$

5. Contigüidad Horaria (no se implementa).

$$costo_5 = 0$$

[b] En [CPU03] esta preferencia es considerada como restricción.

6. Desperdicio de Salón.

$$costo_6 = \sum_{c \in C} \sum_{\substack{s \in S, \\ capacidad(s) > cantalumnos(c)}} Y_{cs}^{(1)} \times (capacidad(s) - cantalumnos(c))$$

7. Espaciamiento en Días.

$$costo_7 = \sum_{\substack{g \in G, \\ tgrupo(g) = teorico}} Y_g^{(5)}$$

8. Similitud de Horario.

$$costo_8 = \sum_{\substack{g \in G, \\ tgrupo(g) = teorico}} \sum_{\substack{c_1 \in C, \\ c_1 \in g}} \sum_{\substack{c_2 \in C, \\ c_1 \neq c_2, \\ c_2 \in g}} Y_{c_1 c_2}^{(6)}$$

3.2.3. Función Objetivo

La función objetivo esta compuesta por la suma del grado de insatisfacción de cada una de las preferencias, ponderadas según su importancia.

$$f = \sum_{i=1}^{8} peso_i \times costo_i$$

donde $costo_i$ corresponde al costo según el grado de insatisfacción de la preferencia $i, i = 1, ..., 8$ y $peso_i$ es el ponderador asociado a la preferencia, el cual le asigna una importancia relativa dentro de la función objetivo f.

3.2.4. Formulación Matemática

$$min \sum_{i=1}^{8} peso_i \times costo_i$$

$$s.a. \sum_{s \in S} \sum_{h \in H} X_{csh} = duracion(c) \qquad \forall c \in C \qquad (R1)$$

$$\sum_{s \in S} Y_{cs}^{(1)} \leq 1 \qquad \forall c \in C \qquad (R2)$$

$$\sum_{c \in C} X_{csh} \leq 1 \qquad \forall s \in S, \forall h \in H \qquad (R3)$$

$$\sum_{s\in S}\sum_{h\in H} X_{csh} \neq 0 \qquad \forall c \in C \qquad (R5)$$

$$hini(c) + duracion(c) - 1 = hfin(c) \qquad \forall c \in C \qquad (R6)$$

$$dia(hini(c)) = dia(hfin(c)) \qquad \forall c \in C \qquad (R7)$$

$$\sum_{\substack{h\in H,\\ h\notin turno(c)}} X_{csh} = 0 \qquad \forall c \in C, \forall s \in S \qquad (R8)$$

$$\sum_{\substack{c_1\in C,\\ grupo(c_1)=g}} \sum_{\substack{c_2\in C,\\ c_1\neq c_2,\\ grupo(c_2)=g}} Y^{(2)}_{c_1c_2} = 0 \qquad \forall g \in G \qquad (R9)$$

donde:

$$costo_1 = \sum_{\substack{c_1\in C,\\ asignatura(grupo(c_1))=a}} \sum_{\substack{c_2\in C,\\ c_1\neq c_2,\\ asignatura(grupo(c_2))=a}} (\sum_{s\in S} X_{c_1sh} \times \sum_{s\in S} X_{c_2sh})$$

$$\forall a \in A / tasignatura(a) \neq electiva \wedge cgrupo(a) = 1, \forall h \in H$$

$$costo_2 = \sum_{c\in C} Y^{(3)}_c$$

$$costo_3 = \sum_{c\in C} \sum_{\substack{s\in S,\\ capacidad(s)<cantalumnos(c)}} Y^{(1)}_{cs} \times (cantalumnos(c) - capacidad(s))$$

$$costo_4 = \sum_{c\in C} Y^{(4)}_c$$

$$costo_5 = 0$$

$$costo_6 = \sum_{c\in C} \sum_{\substack{s\in S,\\ capacidad(s)>cantalumnos(c)}} Y^{(1)}_{cs} \times (capacidad(s) - cantalumnos(c))$$

$$costo_7 = \sum_{\substack{g\in G,\\ tgrupo(g)=teorico}} Y^{(5)}_g$$

$$costo_8 = \sum_{\substack{g\in G,\\ tgrupo(g)=teorico}} \sum_{\substack{c_1\in C,\\ c_1\in g}} \sum_{\substack{c_2\in C,\\ c_1\neq c_2,\\ c_2\in g}} Y^{(6)}_{c_1c_2}$$

Capítulo 4

Adaptación e Implementación de las Metaheurísticas

4.1. Introducción

En la Facultad se están realizando esfuerzos en automatizar el TTP particular de la institución. Dos trabajos anteriores utilizaron dos métodos metaheurísticos para resolver el problema [RC99], [CPU03]. En particular en el proyecto de grado sobre Búsqueda Tabú aplicado al Problema de Asignación de Salones y Horarios [CPU03] se creó el marco adecuado para continuar la investigación sobre metaheurísticas aplicadas al TTP particular. Y poder construir, a partir de ese trabajo, un motor de metaheurísticas para poder incorporar a la herramienta (software) de apoyo a la asignación, que semestralmente realiza Bedelía de la Facultad de Ingeniería.

En el apéndice A se presenta la descripción detallada de Algoritmos Genéticos, Búsqueda Tabú y Optimización por Colonia de Hormigas (los algoritmos Ant System, Ant Colony System y $\mathcal{MAX} - \mathcal{MIN}$ Ant System), las cuales fueron seleccionadas para formar parte del motor, y en éste capítulo se presenta la adaptación de esas metaheurísticas al problema particular de la Facultad (capítulo 3) y la calibración de los parámetros de las mismas.

Este capítulo está organizado de la siguiente forma: primero se describe el entorno de trabajo (hardware y software utilizados), luego se presentan los problemas reales utilizados en el motor, la manera de codificar el problema y finalmente la adaptación de las metaheurísticas al PASHA.

4.2. Entorno de Trabajo

Este trabajo toma como punto de partida el proyecto de grado sobre Tabú Search aplicado al Problema de Asignación de Salones y Horarios [CPU03], el cual fue implementado utilizando Java como lenguaje de programación y MySQL como manejador de base de datos. En ese trabajo, Cuevas, Panzera y Ugalde crearon una base de datos con toda la información de Carreras, Años, Semestres, Asignaturas, Grupos, Clases, Turnos, Salones, Recursos y Horarios del PASHA y además adaptaron e implementaron Búsqueda Tabú para resolver el problema.

Partiendo de lo anterior, en este trabajo la implementación se realiza en Java utilizando MySQL como base de datos.

Características del entorno de trabajo:

- Eclipse Platform, Version: 2.1.2 Build id: 200311030802
- Java 2 SDK, Standard Edition v1.4.2_03
- MySQL Server and Clients 4.0.17
- MySQL Connector/J 3.0.8
- Microsoft Windows XP Professional Version 2002 Service Pack 1
- Intel Pentium 4, CPU 2.66 GHz, RAM 512 MB

Dado que el trabajo se basa en el proyecto de grado, la elección de la plataforma ya estaba determinada al comienzo del trabajo y como es comprobado que la programación con Java tiene menor performance (en tiempo de ejecución) que utilizar C++ por ejemplo, se decide realizar algunos testeos de ejecución en dos sistemas operativos diferentes (Windows XP y SuSE 9.0) sin que se apreciara mejoras dependiendo del sistema operativo utilizado. Al ser Java un lenguaje interpretado se busca diferentes alternativas de compilar el código Java a código nativo de máquina de manera de tratar de acelerar la ejecución. Para eso se utiliza el software Excelsior JET[a] y la opción JIT Compiler brindado por SUN en el compilador de Java. En este caso tampoco se encuentran diferencias sustanciales por lo que se opta por la opción nativa del compilador, que permite pre-compilar el código Java a código de máquina antes de ejecutarlo.

[a]Excelsior JET - http://www.excelsior-usa.com/jet.html

4.3. Problemas Reales

Los problemas que resuelven o intentan resolver las diferentes metaheurísticas implementadas corresponden a problemas reales de asignación de clases de la Facultad, obtenidos de la información que dispone la bedelía de la Facultad ingresados en la base de datos que se utiliza para resolver las asignaciones utilizando el software implementado en el proyecto de grado sobre Tabú Search en el Problema de Asignación de Salones y Horarios [CPU03].

La Facultad cuenta con 40 salones, las clases se dictan de lunes a viernes desde las 8:00 a las 23:00 horas y los sábados desde las 8:00 a las 12:00 horas y como consecuencia se cuenta con 158 medias horas disponibles para dar las clases por semana, por lo que existen 6320 salones x horarios disponibles para asignar. De la información brindada por bedelía se obtienen datos del segundo semestre del año 2003 y del primer semestre del año 2004. Por lo que los problemas que se utilizan son (cuadro 4.1):

- 20032-1 - Asignación de las clases de primer año (158) de todas las carreras del segundo semestre del año 2003.
- 20032-5 - Asignación de las clases de todos los años (588) de todas las carreras del segundo semestre del año 2003.
- 20041-1 - Asignación de las clases de primer año (183) de todas las carreras del primer semestre del año 2004.
- 20041-5 - Asignación de las clases de todos los años (665) de todas las carreras del primer semestre del año 2004.

	Cantidad		
Problemas	Clases	Salones	Horarios
20032-1	158	40	158
20032-5	588	40	158
20041-1	183	40	158
20041-5	665	40	158

Cuadro 4.1: Problemas Reales.

Los problemas 20032-1 y 20041-1 son problemas menos restrictivos que los 20032-5 y 20041-5, ya que solo asignan las clases de primer año de la Facultad utilizando los mismo recursos de salones y horarios disponibles.

4.4. Codificación de la Solución

En la formalización del problema (sección 3.2) se definieron las variables de decisión:

$$X : C \times S \times H \longrightarrow \{0, 1\}$$

donde

$$X_{csh} = \begin{cases} 1 & \text{si la clase } c \text{ es programada en el salón } s \text{ y la hora } h, c \in C, s \in S, h \in H \\ 0 & \text{sino.} \end{cases}$$

La Facultad cuenta con 40 salones, se dictan clases de lunes a viernes de 8:00 hs. a 23:00 hs. y los días sábados de 8:00 hs. a 12:00 hs. y la duración de las mismas se miden en cantidad de medias horas. Además se tiene aproximadamente un total de 600 clases en cada semestre. Por lo que se tiene:

- Clases $= 1, ..., |C|$ donde $|C| \simeq 600$
- Salones $= 1, ..., |S|$ donde $|S| = 40$
- Horarios $= 1, ..., |H|$ donde $|H| = 158$ (medias horas por semana)

Las variables X se pueden codificar como un array booleano de tres dimensiones, en donde cada dimensión representa las clases, los salones y los horarios. Esta manera de codificar llevada a una estructura de datos en un lenguaje de programación lleva a tener un uso poco efectivo y eficiente de la memoria al tener una matriz de tres dimensiones dispersa, es decir, en donde la mayoría de los valores son "False". La matriz tendría una dimensión de $600*40*158 = 3792000$ bytes (un byte por booleano) $\simeq 3703KB \simeq 3{,}6MB$, y teniendo en cuenta que cada clase debe ser asignada a un único salón y como máximo no dura mas de 5 horas (generalmente las clases duran 1 hora y media o 2 horas) tenemos que, codificando de esta manera (como máximo) solo el 0,1 % ($600B*10 \simeq 5{,}9KB \simeq 0{,}006MB$) de la matriz es utilizada efectivamente para determinar la solución.

Una opción posible es codificar la solución como un array de dos dimensiones, por ejemplo en donde un índice son las clases, el otro los salones y el valor de la matriz indica el horario asignado, cuadro 4.2.

	i	j	$X[i,j]$	Largo Solución
1	Clases	Salones	Horarios	$600 * 40 = 24000 \simeq 48000B \simeq 46{,}9KB$
2	Clases	Horarios	Salones	$600 * 158 = 94800 \simeq 189600B \simeq 185{,}2KB$
3	Salones	Horarios	Clases	$40 * 158 = 6320 \simeq 12640B \simeq 12{,}3KB$

Cuadro 4.2: Codificación mediante array de dos dimensiones.

Otra manera de codificar una solución es mediante una array de una dimensión, por ejemplo en donde el índice del array son las clases y el valor del array representan los salones x horarios asignados, cuadro 4.3. Donde se definen Salones x Horarios $= \{(s,h)/(s,h) \in S \times H \wedge s \in S \wedge h \in H\}$, Clases x Horarios $= \{(c,h)/(c,h) \in C \times H \wedge c \in C \wedge h \in H\}$ y Clases x Salones $= \{(c,s)/(c,s) \in C \times S \wedge c \in C \wedge s \in S\}$.

	i	$X[i]$	Largo Solución
4	Clases	Salones x Horarios	$600 \simeq 1200B \simeq 1{,}17KB$
5	Salones	Clases x Horarios	$40 \simeq 80B \simeq 0{,}08KB$
6	Horarios	Clases x Salones	$158 \simeq 316B \simeq 0{,}3KB$
7	Clases x Salones	Horarios	$600 * 40 = 24000 \simeq 48000B \simeq 46{,}9KB$
8	Clases x Horarios	Salones	$600 * 158 = 94800 \simeq 189600B \simeq 185{,}2KB$
9	Salones x Horarios	Clases	$40 * 158 = 6320 \simeq 12640B \simeq 12{,}3KB$

Cuadro 4.3: Codificación mediante array de una dimensión.

Analizando cada una de las opciones posibles de codificación de la solución se tiene que:

1. Clases, Salones → Horarios, a una clase se le asigna un salón y el horario de inicio de la misma, solo el 2,5 % de la matriz contiene información de la solución ($600 * 1 = 600 \simeq 1200B \simeq 1{,}17KB \Rightarrow 2{,}5\,\%$).

2. Clases, Horarios → Salones, a una clase se le asigna los horarios de dictado y el salón, solo el 6 % contiene información de la solución ($600 * 10 = 6000 \simeq 12000B \simeq 11{,}7KB \Rightarrow 6\,\%$).

3. Salones, Horarios → Clases, a cada salón y horario se le asigna la clase, el 95 % contiene información de la solución ($600 * 10 = 6000 \simeq 12000B \simeq 11{,}7KB \Rightarrow 95\,\%$).

4. Clases → Salones x Horarios, a cada clase se le asigna el salón y horario de comienzo, el 100 % contiene información de la solución ($600 \simeq 1200B \simeq 1{,}17KB$).

5. Salones → Clases x Horarios, no es aplicable para el tipo de problema, ya que a un salón se le asigna varias clases en diferentes horarios.

6. Horarios → Clases x Salones, tampoco es aplicable, ya que se dictan diferentes clases en forma simultánea.

7. Clases x Salones → Horarios, idem 1.
8. Clases x Horarios → Salones, idem 2.
9. Salones x Horarios → Clases, idem 3.

Como consecuencia de lo anterior las opciones 5 y 6 no son aplicables para el tipo de problema estudiado, las opciones 1, 2, 7 y 8 utilizan arrays demasiados grandes y en donde un pequeño porcentaje de ellos es usado para determinar una solución. Por lo que se puede decir que la mejor manera de codificar una solución al PASHA es utilizar una de las opciones 3, 4 o 9.

En el proyecto de grado sobre utilización de Ant System en el Problema de Asignación de Horarios y Salones a Cursos [RC99], los autores realizan un estudio de cual es la mejor manera de codificar la solución para el problema y basan todo su trabajo codificando la solución como una matriz de dos dimensiones $X[s, h] = c$, donde c indica la clase asignada o -1 en caso de que no se haya asignado ninguna clase, opción 3, para el salón s y el horario h.

En el proyecto de grado sobre Tabú Search en el Problema de Asignación de Salones y Horarios [CPU03], los autores representan una solución con un array de clases $X[c] = (s, h)$, donde (s, h) indica el salón de la clase y el horario de comienzo de la misma, opción 4.

La forma de representar una solución en este trabajo se basa en la utilizada por [CPU03], en donde $X : C \rightarrow S \times H$ donde $S \times H = \{(s, h)/(s, h) \in S \times H \wedge s \in S \wedge h \in H\}$ o sea que $X(c) = (s, h)$ indica el salón s y el horario h de inicio de la clase c.

4.5. Restricciones y Preferencias

Para evaluar una solución obtenida, la función objetivo tiene en cuenta si se cumplen o no las restricciones y las preferencias definidas en la formalización del PASHA (sección 3.2), de modo que la violación de una de ellas se mide por un costo que afecta el valor de la solución.

Para el caso de las restricciones, el costo asociado a cada una es:

1. Cada clase tiene una duración determinada (R1), la diferencia entre la duración y lo efectivamente asignado.
2. Una clase tiene un único salón (R2), la cantidad de clases que tienen diferente salón asignado.

3. Un horario y salón se programan a una única clase (R3), la cantidad de parejas salón x horario asignados a mas de una clase.

4. Las clases se dictan dentro de los horarios válidos (R4), es implícita y no afecta el costo.

5. Todas las clases son programadas (R5), la cantidad de clases que no se asignaron.

6. El horario de cada clase es contiguo (R6), la cantidad de 1/2 horas de discontinuidad de las clases.

7. Cada clase se dicta en el mismo día (R7), la cantidad de clases que son asignadas en diferentes días.

8. Las clases de un grupo deben estar en el turno especificado (R8), la cantidad de clases que todo o parte del horario asignado esta fuera del turno especificado.

9. Las clases de un grupo deben dictarse en diferentes días (R9), la cantidad de clases de cada grupo que se dictan el mismo día.

y para las preferencias:

1. Superposición, Las clases de una asignatura básica o obligatoria con un solo grupo no se superponen (P1), la cantidad de 1/2 horas que se superponen esas clases.

2. Preferencia de Horario (P2), la cantidad de clases asignadas fuera del horario de preferencia.

3. Capacidad del Salón (P3), la cantidad de alumnos excedidos de la capacidad del salón para cada clase asignada.

4. Recursos Necesarios (P4), la cantidad de clases asignadas a un salón sin los recursos necesarios.

5. Contigüidad Horaria (P5), esta preferencia por su complejidad de implementación no es tenida en cuenta.

6. Desperdicio de Salón (P6), la cantidad de asientos desperdiciados del salón para cada clase asignada a un salón.

7. Espaciamiento en Días (P7), la cantidad de clases de cada grupo que no cumpla el espaciamiento en la semana.

8. Similitud de Horario (P8), la diferencia de horarios (en 1/2 horas) de las clases asignadas de cada grupo.

Las consideraciones anteriores son tenidas en cuenta para calcular la función objetivo.

4.6. Función Objetivo

La función objetivo para una solución cualquiera es el resultado de la suma de los costos de las restricciones y preferencias ponderados por un peso asociado a cada uno de ellos.

Los pesos asociados utilizados son los mismos que los utilizados en el proyecto de grado sobre Tabú Search en el Problema de Asignación de Salones y Horarios [CPU03], cuadro 4.4 (originalmente determinados con los usuarios del sistema).

	Pesos
R1	100.000
R2	100.000
R3	5.000
R4	0
R5	50.000
R6	100.000
R7	100.000
R8	5.000
R9	100.000
P1	400
P2	4.000
P3	5
P4	50
P5	0
P6	1
P7	150
P8	10

Cuadro 4.4: Pesos asociados a las restricciones y preferencias.

La función objetivo se define como:

$$f = \sum_{i=1}^{9} pesoR_i \times costoR_i + \sum_{i=1}^{8} pesoP_i \times costoP_i$$

4.7. Algoritmos Genéticos

Los GA son una técnica robusta y generalmente pueden ser utilizados para una amplia gama de problemas de optimización combinatoria. En esta sección se presenta la adaptación de los GA al PASHA (sección 3.2).

4.7.1. Codificando el PASHA

Antes de utilizar GA se debe dar la forma de codificar los cromosomas. Para codificar las soluciones al PASHA se utiliza codificación directa con múltiples símbolos (sección A.2.2).

El problema consiste en asignar las clases a los diferentes salones y horarios y la Facultad cuenta con aproximadamente 40 salones, 600 clases y 158 horas de clases semanales (medias horas) (sección 4.4).

Utilizando una representación directa de multiples símbolos, una solución al problema se puede representar con un cromosoma compuesto por genes de números enteros.

	Posición Cromosoma	Valor Gene	Largo Cromosoma
1	Clase	Salón x Horario	600
2	Salón	Clase x Horario	40
3	Horario	Clase x Salón	158
4	Clase x Salón	Horario	24.000
5	Clase x Horario	Salón	94.800
6	Salón x Horario	Clase	6.320

Cuadro 4.5: Codificación de la Solución en un Cromosoma.

En el cuadro 4.5 se presentan las posibles maneras de codificar una solución del problema en un cromosoma de números enteros, en donde las clases, los salones y los horarios pueden ser codificados como posición en el cromosomas i o como valor de los genes x_i.

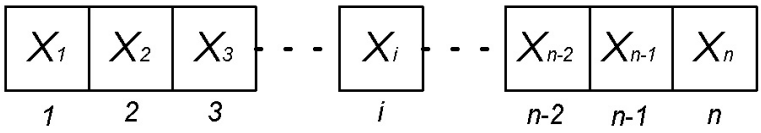


Figura 4.1: Cromosoma para el PASHA.

Las posibles maneras de codificar una solución en un cromosoma son las siguientes (figura 4.1):

1. Clases → Salones x Horarios, la clase i (determinada por la posición i en el cromosoma) se le asigna el salón x horario determinado por x_i. Esta manera de codificar implícitamente cumple la restricción 2 del PASHA (cada clase tiene un único salón).

2. Salones → Clases x Horarios, no es aplicable para esta manera de codificar el cromosoma, ya que a un salón i se le puede asignar mas de una clase x horario x_i por semana.

3. Horarios → Clases x Salones, idem al anterior, se dictan varias clases simultáneamente en varios salones en cada horario i.

4. Clases x Salones → Horarios, a cada clase x salón i se le asigna el horario x_i. Esta codificación necesita de cromosomas muy grandes y que además gran parte del cromosoma no brinda información de la solución.

5. Clases x Horarios → Salones, para cada clase x horario i se le asigna un salón x_i. Idem anterior.

6. Salones x Horarios → Clases, cada salón x horario i en el cromosoma se le asigna una clase x_i. Esta manera de codificar implícitamente cumple la restricción 3 del PASHA (un horario y salón se programan a una única clase). Esta manera de codificar genera cromosomas grandes, que además, como las clases mas largas duran 5 horas (10 1/2 horas), por lo menos la mitad del cromosoma no contiene información de la solución.

Cómo consecuencia, la mejor manera de codificar el PASHA en un cromosoma directo de múltiples símbolos es que la posición en el cromosoma determine la clase y el valor del gen en esa posición el salón y el horario asignado (inicio de la clase).

4.7.2. Aptitud, Población Inicial y Operaciones Genéticas

Función de Aptitud

Generalmente la función de aptitud esta compuesta por la función objetivo del problema y una función de castigo. Para el problema tratado en este trabajo no se considera la función de castigo, ya que la no factibilidad de las soluciones pueden ser castigadas en la función objetivo aumentando el peso del no cumplimiento de cada restricción. Como para este problema, el objetivo es minimizar el costo, la función de aptitud es el inverso de la función objetivo.

Población Inicial

La población inicial puede ser generada en forma aleatoria, para cada clase (determinado por la posición en el cromosoma) se le asigna aleatoriamente un salón x horario cualquiera o puede ser generada mediante la utilización de un procedimiento constructivo (sección 4.10).

Reproducción y Selección

La operación de reproducción selecciona las soluciones de la población anterior para realizarles las operaciones genéticas. En esta adaptación se considera la selección proporcional a la aptitud, por ranking y por torneo. En esta adaptación se elige reemplazar todas las soluciones de la población anterior por los generados en la selección y luego de aplicar las operaciones genéticas.

Cruzamiento

Para realizar el cruzamiento de dos cromosomas de la población se generan en forma aleatoria las posiciones (según el valor del parámetro que indica la cantidad de puntos de cruzamiento) en el cromosoma por donde se realizará el cruzamiento.

Mutación

La mutación en la adaptación de GA consiste en variar el salón x horario asignado a una clase según la probabilidad de mutación para todas las clases de cada cromosoma de la población. La variación se hace simplemente asignando una pareja salón x horario generada en forma aleatoria.

4.7.3. Pseudocódigo GA

La estructura del algoritmo GA para el PASHA se muestra en los algoritmos 1, 2, 3, 4 y 5. Los parámetros que controlan el algoritmo son:

- $tipo_{pi}$, el Tipo de Población Inicial (aleatoria o construida),
- $tamano_p$, el Tamaño de la Población,
- $tiempo_e$, el Tiempo de Ejecución del algoritmo, medido en minutos,
- $tipo_r$, el Tipo de Reproducción (proporcional a la aptitud, ranking o torneo),
- $puntos_c$, la cantidad de Puntos de Cruzamiento,
- $probabilidad_c$, la Probabilidad de Cruzamiento,
- $probabilidad_m$, la Probabilidad de Mutación y
- $probabilidad_t$, la Probabilidad de Torneo a aplicar en la reproducción por torneo.

Algorithm 1 *Metaheurística GA*

1: P=InicializarPoblación($tipo_{pi}, tamano_p$)
2: $S_{mejor\_global} = \emptyset$
3: $t = 0$ {tiempo de ejecución}
4: **while** $t \leq tiempo_e$ **do**
5: Q=ReproducciónPoblación($P, tipo_r, probabilidad_t$)
6: Q=CruzamientoPoblación($Q, puntos_c, probabilidad_c$)
7: Q=MutaciónPoblación($Q, probabilidad_m$)
8: EvaluarPoblación(Q)
9: P = ActualizarPoblación($Q, tamano_p$)
10: S_{mejor_global}= MejorSolución(S_{mejor_global},MejorPoblación(P))
11: $i++$ {incrementar el tiempo de ejecución}
12: **end while**

Algorithm 2 *InicializarPoblación($tipo_{pi}, tamano_p$)*

1: **if** $tipo_{pi}$=Aleatoria **then**
2: P=InicializarPoblaciónAleatoria($tamano_p$)
3: **end if**
4: **if** $tipo_{pi}$=Construida **then**
5: P=InicializarPoblaciónConstruida($tamano_p$)
6: **end if**

Algorithm 3 *ReproducciónPoblación($P, tipo_r, probabilidad_t$)*

1: **if** $tipo_r$=Proporcional a la Aptitud **then**
2: Q=ReproducciónProporcionalAptitud(P)
3: **end if**
4: **if** $tipo_r$=Ranking **then**
5: Q=ReproducciónRanking(P)
6: **end if**
7: **if** $tipo_r$=Torneo **then**
8: Q=ReproducciónTorneo($probabilidad_t$)
9: **end if**

Algorithm 4 *CruzamientoPoblación($Q, puntos_c, probabilidad_c$)*

1: **for all** $(q_1, q_2) \in Q \times Q \wedge$ Probabilidad($probabilidad_c$) **do**
2: $Q = Q \bigcup$ Cruzamiento($q_1, q_2, puntos_c$)
3: **end for**

Algorithm 5 *MutaciónPoblación($Q, probabilidad_m$)*

1: **for all** $(q) \in Q$ **do**
2: **for all** $(c) \in q \wedge$ Probabilidad($probabilidad_m$) **do**
3: $c =$ Mutación(c)
4: **end for**
5: **end for**

4.8. Búsqueda Tabú

El TS es utilizado en una una amplia gama de problemas de optimización combinatoria. En esta sección se presenta la adaptación del TS al PASHA formulado en el capítulo 3.

La adaptación de TS al PASHA se basa en la realizada en el proyecto de grado sobre Tabú Search en el Problema de Asignación de Salones y Horarios [CPU03]. A la adaptación e implementación realizada en el proyecto de grado, para este trabajo, se mejoró el algoritmo de cálculo de la función objetivo (el mismo es utilizado también por las otras metaheurísticas) de manera de hacerlo más eficiente en tiempo de ejecución.

4.8.1. Definición del Vecindario y Movimientos Vecinos

Un movimiento vecino a una solución S cualquiera es la asignación a una clase de un salón x horario distinto al actual, de manera que el vecindario de S se define como $\mathcal{N}(S) = \{S'/S' \text{ difiere de } S \text{ en un movimiento}\}$.

En el proyecto de grado [CPU03] al construir el conjunto de vecinos de una solución se tiene en cuenta que el movimiento genere una solución que cumpla las restricciones 1 y 7[b]. El método implementado permite realizar movimientos que generan soluciones no factibles (solo controlan las restricciones 1 y 7) y los autores fundamentan esta elección, basado en el estudio realizado de trabajos anteriores, en el sentido de evitar que el algoritmo quede estancado en un mínimo local y que no pueda salir de él al no encontrar soluciones vecinas factibles. De esta manera logran interconectar el espacio se soluciones factibles.

4.8.2. Lista Tabú

En la implementación de TS para el PASHA se consideran dos listas tabú:

- Lista Tabú Clase
- Lista Tabú Salón Horario

En la Lista Tabú Clase se guardan las clases que son asignadas en cada movimiento y en la Lista Tabú Salón Horario se guarda la clase y el salón x horario asignado al mismo en cada movimiento.

[b]Cada clase tiene una duración determinada y cada clase se dicta en el mismo día.

Cuando la clase c se mueve a un salón x horario (s, h) se agrega c en la Lista Tabú Clase y $(c, (s, h))$ en la Lista Tabú Salón Horario. Esto significa que durante cierto tiempo la clase c no se puede mover y que durante otro tiempo no se puede mover a (s, h). La implementación de la segunda lista tabú tiene sentido siempre que sea mas larga que la Lista Tabú Clase.

Cuando el número de clases en conflicto es pequeño (menos de un 7 % de la cantidad de clases que se manejan), la primer lista tabú puede prohibir los movimientos que llevan a una solución vecina adecuada. En este caso, no se tiene en cuenta el status de tabú de un movimiento si es generado por la primer lista.

4.8.3. Criterio de Aspiración

El criterio de aspiración indica el valor al que se aspira llegar de la función objetivo desde una solución S. Para el caso del PASHA se define la función de aspiración como:

$$A(f(S)) = f(S_{mejor\_global})$$

O sea que un movimiento que esté en las listas tabú solo es aceptado si mejora el valor de la función objetivo de la mejor solución obtenida hasta ese momento.

4.8.4. Pseudocódigo TS

La estructura del algoritmo TS para el PASHA se muestra en el algoritmo 6. Los parámetros que permiten controlar el algoritmo son:

- $tiempo_e$, el Tiempo de Ejecución,
- $tipo_{si}$, el Tipo de Solución Inicial,
- $cantidad_{sv}$, la Cantidad de Soluciones Vecinas V,
- $largo_{ltc}$, el largo de la Lista Tabú Clase L_c y
- $largo_{ltsh}$, el largo de la Lista Tabú Salón Horario L_{sh}.

El procedimiento *InicializarSolución* genera la solución inicial, que puede ser una solución generada en forma aleatoria o mediante la utilización de un método constructivo. *GenerarVecinos* construye un subconjunto del vecindario de S_{actual} ($V \subseteq \mathcal{N}(S_{actual})$) y *ElegirMejorVecino* elige el mejor movimiento de V que no esté en la listas tabú o que cumpla el criterio de aspiración. Si $v \neq \emptyset$ aplica el movimiento en S_{actual} y se actualizan las listas tabú y la mejor solución global. El procedimiento termina la ejecución cuando llegue al tiempo máximo determinado.

Algorithm 6 *Metaheurística TS*

1: S_{actual} = InicializarSolución($tipo_{si}$)
2: $L_c = \emptyset$ {Lista Tabú Clase}
3: $L_{sh} = \emptyset$ {Lista Tabú Salón Horario}
4: $S_{mejor\_global} = S_{actual}$
5: $t = 0$ {tiempo de ejecución}
6: **while** $t \leq tiempo_e$ **do**
7: V = GenerarVecinos($\mathcal{N}(S_{actual}), cantidad_{sv}$)
8: v = ElegirMejorVecino(V, L_c, L_{sh})
9: S_{actual} = AplicarMovimiento(v)
10: $L_c = L_c \bigcup$ Clase(v)
11: $L_{sh} = L_{sh} \bigcup$ SalonHorario(v)
12: $S_{mejor\_global} = Mejor(S_{actual}, S_{mejor\_global})$
13: $t++$ {incrementar el tiempo de ejecución}
14: **end while**

4.9. Optimización por Colonia de Hormigas

Los tipos de problemas que se pueden resolver con los algoritmos ACO son los que se pueden representar como un grafo ponderado $G = (N, A)$, donde A es el conjunto de aristas que conectan los nodos N (ver apéndice A). En esta sección se presenta la adaptación de los algoritmos ACO al PASHA formulado en el capítulo 3.

Para el caso del PASHA se tiene que el conjunto de nodos del grafo es $N = C \bigcup (S \times H)$, o sea $N = \{c \in C\} \bigcup \{(s,h)/(s,h) \in S \times H \wedge s \in S \wedge h \in H\}$ y $A = \{a_{rs}/r \in C \wedge s \in S \times H\}$ en donde A es el conjunto de aristas que unen las clases con las parejas salón x horario, es decir $A = \{(c,(s,h))/c \in C \wedge (s,h) \in S \times H \wedge s \in S \wedge h \in H\}$.

El conjunto de soluciones factibles S que cumplen las restricciones Ω, está incluido en el conjunto potencia de A, $S \subset Pot(A)$. El problema es encontrar el $s_{opt} \in S$ que minimice la función de costo C.

Existen dos formas de que las hormigas construyan una solución en el grafo $G = (N, A)$ donde $N = C \bigcup (S \times H)$, una es que las hormigas recorran las clases y para cada una de ellas intenten asignar una pareja salón x horario según la regla de transición, y la otra opción es que cada hormiga recorra los salones x horarios e intente asignar una clase.

En el proyecto de grado sobre utilización de Ant System en el Problema de Asignación de Horarios y Salones a Cursos [RC99], los autores justifican el uso de la segunda opción, o sea que cada hormiga recorra los salones x horarios y les asigne una clase. Pero con esta opción, tuvieron que modificar el comportamiento básico del algoritmo para evitar que las hormigas asignen a los primeros salones x horarios todas las clases (la regla de transición indica que clase asignar, pero no si es mejor para la solución

global no asignarle ninguna clase a una pareja salón x horario) controlando la codicia del método. La mejor respuesta de si asignar o no una clase debería responderse en el contexto de la solución global construida y se considera que mientras se construye la solución, esa respuesta no se puede contestar de la mejor forma. Por ésta razón es que en la adaptación de los algoritmos ACO al PASHA se considera la opción de recorrer las clases y asignarles los salones x horarios.

4.9.1. Visibilidad

Generalmente la visibilidad se define como el beneficio de elegir cierto camino, dado por la información heurística. En el caso del PASHA esa información está dada por el costo asociado a cada posible elección y el costo está dado por el cumplimiento de las restricciones y preferencias. De la manera que las hormigas construyen su solución, solo generan soluciones factibles, o sea que las soluciones construidas cumplen implícitamente las restricciones. La mejor manera de medir la visibilidad de una hormiga posicionada en una clase es calcular el valor total de las preferencias teniendo en cuenta la solución construida hasta el momento (de las clases visitadas con anterioridad por la misma) y las posibles asignaciones de salones x horarios. De esta manera la visibilidad queda definida por:

$$\eta_{csh} = 1/(1 + \sum_{i=1}^{8} pesoP_i \times costoP_i)$$

Las preferencias se separan en dos grupos: locales y globales. Las locales son aquellas que su valor no depende de la asignación, sino que directamente el valor es calculado para cada par (clase, (salón, horario)) (no depende de los pasos anteriores o posteriores de la hormiga. Y las globales son las que su valor depende de la asignación construida hasta ese momento por la hormiga. Las locales se pueden calcular previamente y no varían durante la ejecución de los algoritmos, sin embargo, las globales se tienen que calcular en cada paso. Las preferencias locales son las preferencias 3, 4 y 5, y las otras son preferencias globales.

4.9.2. Actualización del Rastro

El rastro dejado por una hormiga puede reflejarse en las aristas del grafo básicamente en dos formas: cada vez que la hormiga recorre una arista se actualiza el rastro de esa arista inmediatamente y el rastro de todas las aristas recorridas por la hormiga se actualiza luego que la misma terminó su recorrido.

En su forma mas simple, el rastro sólo indica que una arista fue recorrida, no importando que tan bueno haya sido elegir esa arista o que tan buena resulte ser la

solución construida. Es decir, en este sentido todos los caminos tomados tienen la misma bondad.

Una alternativa posible es obtener la cantidad de rastro a actualizar en las aristas según que tan buena sea la solución encontrada por la hormiga. En este caso, la mejor o peor solución es determinada por el valor de la función objetivo.

La actualización del rastro, elegida en este trabajo, para cada arista visitada es la siguiente:

$$1/C(S)$$

donde $C(S)$ es el valor de la función objetivo para la solución S construida por la hormiga.

4.9.3. Pseudocódigo AS

La estructura del algoritmo AS para el PASHA se muestra en los algoritmos 7, 8, 9 y 10. Los parámetros que controlan el algoritmo son:

- $tiempo_c$, el Tiempo de Ejecución del algoritmo, medido en minutos,
- m, la Cantidad de Hormigas utilizadas en cada iteración,
- τ_0, el Rastro Inicial para cada clase, salón y horario,
- ρ, la Evaporación del rastro,
- α, la Ponderación del Rastro τ, y
- β, la Ponderación de la Visibilidad η.

Algorithm 7 *Metaheurística AS*

InicializarParámetros()
$S_{mejor\_global} = \emptyset$
$t = 0$ {tiempo de ejecución}
for all $c \in C, s \in S, h \in H$ **do**
  $\tau_{csh} = \tau_0$
end for
while $t \leq tiempo_e$ **do**
  GeneraciónActividadHormigas()
  EvaporaciónFeromona()
  $i++$ {incrementar el tiempo de ejecución}
end while

Algorithm 8 *GeneraciónActividadHormigas()*

for $k = 1$ **to** m {número de hormigas} **do**
  NuevaHormiga(k)
end for

Algorithm 9 *NuevaHormiga(k)*

$L^k = \emptyset$ {lista tabú de (s,h) asignados por la hormiga k}
$S^k = \emptyset$ {solución construida por la hormiga k}
for all $c \in C$ {cada clase en orden decreciente en importancia} **do**
  $\mathcal{N}_c^k$=ConstruirConjuntoVecinos(c,L^k,S^k) {$(s,h) \notin L^k$ y $S^k \bigcup (c,s,h)$ cumple restricciones}
  for all $(s,h) \in \mathcal{N}_c^k$ **do**
    $p_{csh}^k = \frac{[\tau_{csh}]^\alpha \cdot [\eta_{csh}]^\beta}{\sum_{(t,i) \in \mathcal{N}_c^k} [\tau_{cti}]^\alpha \cdot [\eta_{cti}]^\beta}$
  end for
  {(s,h)} = AplicarDecisiónHormiga($c, P, \mathcal{N}_c^k$)
  $S^k = S^k \bigcup \{(c,s,h)/(s,h) \in \{(s,h)\}\}$
  $L^k = L^k \bigcup \{(s,h)\}$
end for
S_{mejor_global}= MejorSolución(S_{mejor_global}, S^k)

Algorithm 10 *EvaporaciónFeromona()*

for all (c,s,h) **do**
  $\tau_{csh} = (1-\rho).\tau_{csh}$
end for
for $k = 1$ **to** m {número de hormigas} **do**
  for all $(c,s,h) \in S^k$ **do**
    $\tau_{csh} = \tau_{csh} + 1/C(S^k)$
  end for
end for

4.9.4. Pseudocódigo ACS

La estructura del algoritmo ACS para el PASHA se muestra en los algoritmos 11, 12, 13 y 14. Los parámetros que controlan el algoritmo son:

- $tiempo_c$, el Tiempo de Ejecución del algoritmo, medido en minutos,
- m, la Cantidad de Hormigas utilizadas en cada iteración,
- τ_0, el Rastro Inicial para cada clase, salón y horario,
- ρ, la Evaporación del rastro,
- q_0, la Probabilidad q_0, y
- β, la Ponderación de la Visibilidad η.

Algorithm 11 *Metaheurística ACS*

```
1: InicializarParámetros()
2: S_mejor_global = ∅
3: t = 0 {tiempo de ejecución}
4: for all c ∈ C, s ∈ S, h ∈ H do
5:     τ_csh = τ_0
6: end for
7: while t ≤ tiempo_e do
8:     GeneraciónActividadHormigas()
9:     AccionesGlobales()
10:    i++ {incrementar el tiempo de ejecución}
11: end while
```

Algorithm 12 *GeneraciónActividadHormigas()*

```
1: for k = 1 to m {número de hormigas} do
2:     NuevaHormiga(k)
3: end for
```

Algorithm 13 *NuevaHormiga(k)*

$L^k = \emptyset$ {lista tabú de (s,h) asignados por la hormiga k}
$S^k = \emptyset$ {solución construida por la hormiga k}
for all $c \in C$ {cada clase en orden decreciente en importancia} **do**
  $\mathcal{N}_c^k$=ConstruirConjuntoVecinos(c,L^k,S^k) {$(s,h) \notin L^k$ y $S^k \bigcup (c,s,h)$ cumple restricciones}
  for all $(s,h) \in \mathcal{N}_c^k$ **do**
    $b_{csh} = \tau_{csh}.\eta_{csh}^{\beta}$
  end for
  q = GenerarValorRandómico$(0,1)$
  if $q \leq q_0$ **then**
    {(s,h)} = max$(b_{csh}, \mathcal{N}_c^k)$
  else
    for all $(s,h) \in \mathcal{N}_c^k$ **do**
      $p_{csh}^k = \frac{b_{chs}}{\sum_{(t,i)\in\mathcal{N}_c^k} b_{cti}}$
    end for
    {(s,h)} = AplicarDecisiónHormiga$(c, P, \mathcal{N}_c^k)$
  end if
  $S^k = S^k \bigcup \{(c,s,h)/(s,h) \in \{(s,h)\}\}$
  $L^k = L^k \bigcup \{(s,h)\}$
  $\tau_{rs} = (1-\varphi).\tau_{rs} + \varphi.\tau_0$
end for

Algorithm 14 *AccionesGlobales()*

for all S^k **do**
  BúsquedaLocal(S^k) {opcional}
end for
S_{mejor_actual}= MejorSolución(S^k)
if Mejor($S_{mejor\_actual}, S_{mejor\_global}$) **then**
  $S_{mejor\_global} = S_{mejor\_actual}$
end if
for all $(c,s,h) \in S_{mejor\_global}$ **do**
  {el procedimiento EvaporaciónFeromona() es sustituido por lo siguiente}
  $\tau_{csh} = (1-\rho).\tau_{csh} + \rho/C(S_{mejor\_global})$
end for

4.9.5. Pseudocódigo $\mathcal{MM}$AS

La estructura del algoritmo $\mathcal{MM}$AS para el PASHA se muestra en los algoritmos 15, 16, 17 y 18. Los parámetros que controlan el algoritmo son:

- $tiempo_c$, el Tiempo de Ejecución del algoritmo, medido en minutos,
- m, la Cantidad de Hormigas utilizadas en cada iteración,

- τ_{min}, el Rastro Mínimo,
- τ_{max}, el Rastro Máximo,
- ρ, la Evaporación del rastro,
- α, la Ponderación del Rastro τ, y
- β, la Ponderación de la Visibilidad η.

Algorithm 15 *Metaheurística $\mathcal{MMAS}$*

1: InicializarParámetros()
2: $S_{mejor\_global} = \emptyset$
3: $t = 0$ {tiempo de ejecución}
4: **for all** $c \in C, s \in S, h \in H$ **do**
5: $\tau_{csh} = \tau_0$
6: **end for**
7: **while** $t \leq tiempo_e$ **do**
8: GeneraciónActividadHormigas()
9: AccionesGlobales()
10: $i++$ {incrementar el tiempo de ejecución}
11: **end while**

Algorithm 16 *GeneraciónActividadHormigas()*

1: **for** $k = 1$ **to** m {número de hormigas} **do**
2: NuevaHormiga(k)
3: **end for**

Algorithm 17 *NuevaHormiga(k)*

1: $L^k = \emptyset$ {lista tabú de (s, h) asignados por la hormiga k}
2: $S^k = \emptyset$ {solución construida por la hormiga k}
3: **for all** $c \in C$ {cada clase en orden decreciente en importancia} **do**
4: $\mathcal{N}_c^k$=ConstruirConjuntoVecinos(c,L^k,S^k) {$(s, h) \notin L^k$ y $S^k \bigcup (c, s, h)$ cumple restricciones}
5: **for all** $(s, h) \in \mathcal{N}_c^k$ **do**
6: $p_{csh}^k = \frac{[\tau_{csh}]^\alpha \cdot [\eta_{csh}]^\beta}{\sum_{(t,i)\in\mathcal{N}_c^k} [\tau_{cti}]^\alpha \cdot [\eta_{cti}]^\beta}$
7: **end for**
8: {(s,h)} = AplicarDecisiónHormiga($c, P, \mathcal{N}_c^k$)
9: $S^k = S^k \bigcup \{(c, s, h)/(s, h) \in \{(s, h)\}\}$
10: $L^k = L^k \bigcup \{(s, h)\}$
11: **end for**

Algorithm 18 *AccionesGlobales()*

for all S^k **do**
  BúsquedaLocal(S^k) {opcional}
end for
S_{mejor_actual}= MejorSolución(S^k)
if Mejor($S_{mejor\_actual}, S_{mejor\_global}$) **then**
  $S_{mejor\_global} = S_{mejor\_actual}$
end if
S_{mejor}= Decisión($S_{mejor\_actual}, S_{mejor\_global}$)
{el procedimiento EvaporaciónFeromona() es sustituido por lo siguiente}
for all (c, s, h) **do**
  $\tau_{csh} = (1 - \rho).\tau_{csh}$
end for
for all $(c, s, h) \in S_{mejor}$ **do**
  $\tau_{csh} = \tau_{csh} + 1/C(S_{mejor})$
end for
for all (c, s, h) **do**
  if $\tau_{csh} < \tau_{min}$ **then**
    $\tau_{csh} = \tau_{min}$
  end if
  if $\tau_{csh} > \tau_{max}$ **then**
    $\tau_{csh} = \tau_{max}$
  end if
end for

4.10. Soluciones Iniciales

Tanto TS como GA utilizan como punto de partida una o mas soluciones al problema. Para ambos casos las soluciones iniciales utilizadas pueden ser generadas en forma aleatoria o mediante la utilización de un método constructivo goloso (greedy).

El método utilizado para construir soluciones iniciales se basa en la forma que utiliza una hormiga en ACO para construir una solución, solo que en este caso la hormiga no tiene en cuenta el rastro sino que solo maneja la visibilidad para decidir que nodo recorrer en el proceso de construcción. En el algoritmo 19 se muestra como se construyen las soluciones iniciales.

En la generación de soluciones iniciales en forma aleatoria simplemente para cada clase se le asigna un salón x horario en forma aleatoria.

En el caso de soluciones construidas, las mismas son soluciones factibles que cumplen todas las restricciones y las generadas en forma aleatoria generalmente no son soluciones factibles. En todos los casos las soluciones iniciales construidas por el algoritmo 19 son de mejor costo que las soluciones manuales construidas en bedelía y las obtenidas en el proyecto de grado [CPU03].

Algorithm 19 *SoluciónInicial*

1: $L = \emptyset$ {lista tabú de (s, h) asignados}
2: $S = \emptyset$ {solución construida}
3: **for all** $c \in C$ {cada clase en orden decreciente en importancia} **do**
4: $\mathcal{N}_c$=ConstruirConjuntoVecinos(c, L, S) {$(s, h) \notin L$ y $S \bigcup (c, s, h)$ cumple restricciones}
5: **for all** $(s, h) \in \mathcal{N}_c$ **do**
6: $p_{csh} = \frac{\eta_{csh}}{\sum_{(t,i)\in\mathcal{N}_c} \eta_{cti}}$
7: **end for**
8: {(s,h)} = AplicarDecisiónHormiga$(c, P, \mathcal{N}_c)$
9: $S = S \bigcup \{(c, s, h)/(s, h) \in \{(s, h)\}\}$
10: $L = L \bigcup \{(s, h)\}$
11: **end for**

4.11. Calibración de las Metaheurísticas

Desde siempre en la utilización de metaheurísticas para resolver los problemas de optimización, los investigadores se han encontrado que una parte muy importante de sus investigaciones y la que determina en buena medida el buen desempeño de una metaheurística es el de encontrar el mejor seteo de valores de los parámetros que controlan los algoritmos. A pesar de la importancia del problema, poco se ha podido avanzar en este tema. Ciertamente existen guías que pretenden ayudar al investigador a encontrar los mejores valores. Pero todavía hoy en día, la mejor o peor calibración de los parámetros queda basado en la intuición y experiencia del investigador, basándose en una metodología de ensayo y error. Sin embargo, en los últimos años, se están empezando a utilizar herramientas de simulación en la calibración de las metaheurísticas, uno de ellas es la utilización de análisis factorial [LK91], pero no se encuentra útil su aplicación en este trabajo, por lo que el método de calibración se basa en la observación del comportamiento de los diferentes algoritmos al variar los valores de los parámetros.

4.12. Determinación de Valores de los Parámetros

Para determinar los mejores valores de cada parámetro para cada metaheurística adaptada e implementada para resolver el PASHA se realiza una serie de ejecuciones de los algoritmos variando los valores de los diferentes parámetros (ver apéndice B). Para cada variación de los parámetros se realizaron 5 corridas de 30', tomando siempre las mismas soluciones iniciales (para TS y GA) construidas y aleatorias generadas antes de realizar la calibración (ver sección B.2). En el siguiente cuadro (4.6) se muestra el mejor resultado obtenido en todas las ejecuciones para cada metaheurística y cada problema (20032-5 y 20041-5 definidos en la sección 4.3) y el mejor promedio

obtenido para cada una de las 5 ejecuciones variando cada parámetro.

	Problemas			
	20032-5		20041-5	
Metaheurística	mejor	promedio	mejor	promedio
GA	563762	597403.20	118992	127478.6
TS	240178	258416.20	59594	65486
AS	223544	229812.2	35913	39396.8
ACS	229649	234325.60	29664	31543.40
$\mathcal{MM}$AS	219444	226214.20	36278	38769

Cuadro 4.6: Mejores costos obtenidos (ejecuciones de 30' realizadas en la calibración).

Luego de un extenso estudio y de experimentación se puede concluir que los mejores resultados se obtienen para:

- Algoritmos Genéticos (algoritmo 1) fijando:
  - el Tipo de Población Inicial en Construida, con las soluciones iniciales construidas tienen mejor costo que las soluciones iniciales aleatorias, lo que hace que el algoritmo se comporte mejor en los tiempos utilizados para la calibración.
  - el Tipo de Reproducción en Ranking, sin embargo con reproducción por Torneo con una probabilidad de Torneo en el entorno de 0.2 se obtienen resultados similares.
  - el Tamaño de la Población en 20 individuos, al aumentar el tamaño de la población el algoritmo converge mas lento a buenas soluciones, estando la performance muy atada al tiempo de ejecución.
  - la cantidad de Puntos de Cruzamiento entre 50 y 90, sin embargo no se observan grandes diferencias entre los diferentes valores del parámetro.
  - la Probabilidad de Cruzamiento entre 0.6 y 0.9, no existen diferencias grandes entre los diferentes valores del parámetro.
  - la Probabilidad de Mutación en 0.005 y
  - la Probabilidad de Torneo en 0.2, si se utiliza Tipo de Reproducción por Torneo.
- Tabú Search (algoritmo 6) fijando:
  - el Tipo de Solución Inicial en Construida, idem que para Algoritmos Genéticos.
  - la Cantidad de Soluciones Vecinas en 500,
  - el Largo de la Lista Tabú Clase entre 30 y 100 y

    - el Largo de la Lista Tabú Salón Horario entre 30 y 200 (el largo de esta lista debe ser mayor que el Largo de la Lista Tabú Clase).

- Ant System (algoritmo 7) fijando:
    - la Cantidad de Hormigas entre 2 y 40,
    - el Rastro Inicial entre 1 y 50, no se observan diferencias de resultado dependiendo del valor inicial del rastro
    - la Evaporación entre 0.0005 y 0.001,
    - la Ponderación del Rastro entre 0.01 y 10 y
    - la Ponderación de la Visibilidad entre 20 y 30.
- Ant Colony System (algoritmo 11) fijando:
    - la Cantidad de Hormigas entre 5 y 10,
    - el Rastro Inicial entre 2 y 50,
    - la Evaporación en 0.05,
    - la Evaporación Local entre 0.0005 y 0.3,
    - la Probabilidad q0 entre 0.7 y 0.9 y
    - la Ponderación de la Visibilidad entre 30 y 50.
- $\mathcal{MAX} - \mathcal{MIN}$ Ant System (algoritmo 15) fijando:
    - la Cantidad de Hormigas entre 10 y 30,
    - el Rastro Mínimo entre 0.2 y 20,
    - el Rastro Máximo entre 1 y 100,
    - la Evaporación entre 0.01 y 0.05,
    - la Ponderación del Rastro en 5 y 10 y
    - la Ponderación de la Visibilidad en 50.

4.13. Conclusiones sobre la Calibración de las Metaheurísticas

Algunas conclusiones que se pueden deducir de la calibración de parámetros para las diferentes metaheurísticas implementadas son dadas a continuación:

- Los resultados obtenidos con soluciones iniciales construidas siempre son de menor costo que los obtenidos con soluciones aleatorias, en los tiempos de ejecución manejados en la calibración.

- Los parámetros son robustos, ya que el comportamiento es similar para los diferentes problemas.
- Corridas con los mismos valores de los parámetros devuelven resultados similares.
- Además, una vez fijados los diferentes valores "razonables"que pueden tomar los parámetros (paso 3 de la calibración), éstos no presentan gran sensibilidad a los cambios de sus respectivos valores.

Estas consideraciones son válidas para todas las metaheurísticas consideradas en este trabajo.

Capítulo 5

Resultados Obtenidos

5.1. Introducción

En el capítulo anterior se adaptaron las diferentes metaheurísticas para poder resolver el PASHA y se realizó una calibración de los parámetros de las mismas. En este capítulo se muestran los resultados obtenidos con las mismas, de manera de determinar, en lo posible, una recomendación de cuál de las metaheurísticas es la mejor en términos de la calidad de la solución obtenida, medida en costo, así como en la velocidad de convergencia a buenas soluciones.

En las siguientes secciones se describen los "problemas reales" utilizados en la comparación, la forma de comparación utilizada, los valores tomados de los parámetros y finalmente los resultados obtenidos.

5.2. Datos Reales del PASHA

Como resultado de los trabajos de grado anteriores [RC99] y [CPU03], en la actualidad en el sistema de apoyo realizado para la Bedelía de la Facultad se cuenta con datos reales del segundo semestre del año 2003 y del primer semestre del año 2004. Teniendo en cuenta que para un mismo semestre pero en diferentes años lectivos la realidad del problema no varía en mayor medida, se puede considerar que con los datos de los dos semestres (aunque de diferentes años) se tiene la información completa de la realidad de la asignación de clases en la Facultad para cada año.

Analizando los datos de las dos instancias del problema, se observa que para el primer semestre del 2004 los datos que determinan la preferencia 2[a] no están completos y si

[a]Preferencia de Horario: Todas las clases se programan dentro del horario de preferencia especificado.

están completos para el segundo semestre del 2003. Por lo que al tener incompleta una de las instancias para el problema definido hace que el problema original varíe su definición según los datos reales que se tiene. Como consecuencia no es el mismo problema para el segundo semestre del 2003 que para el primer semestre del 2004.

En el primer semestre del 2004 de 665 clases solo 24 clases tiene especificado un horario de preferencia y para el segundo semestre del 2003 de un total de 588 clases, 368 clases tienen especificado la preferencia de horarios. El resto de las clases tienen como preferencia de horario todo el día.

Al tener una instancia del problema con los datos incompletos según la definición formal del PASHA, lleva a transformar el problema original a un nuevo problema que no considere esa preferencia.

Por lo que originalmente se tienen dos instancias del problema:

- 20032-5 - Asignación de las clases de todos los años (588) de todas las carreras del segundo semestre del año 2003.
- 20041-5 - Asignación de las clases de todos los años (665) de todas las carreras del primer semestre del año 2004.

Y para el nuevo problema sin considerar la preferencia 2:

- 20032-5s/P2 - Asignación de las clases de todos los años (588) de todas las carreras del segundo semestre del año 2003 pero sin considerar preferencia 2.
- 20041-5s/P2 - Asignación de las clases de todos los años (665) de todas las carreras del primer semestre del año 2004 pero sin considerar preferencia 2.

5.3. Plan de Experimentación

Para obtener los resultados del comportamiento de las diferentes metaheurísticas resolviendo el PASHA, primero se toman los mejores valores de los parámetros obtenidos en el análisis de sensibilidad de los mismos (sección 4.12 y apéndice B), luego con estos valores de los diferentes parámetros, se toman los problemas 20032-5 y 20041-5 (sección 4.3) y se realiza una corrida de 12 horas (720 minutos) para cada una de las metaheurísticas y cada uno de los problemas, teniendo un tiempo total de ejecución de 120 horas ($5 * 2 * 720' = 7200' = 120hs$). Debido a los resultados obtenidos en la calibración (apéndice B) en donde las corridas con los mismos valores de los

parámetros devuelven resultados similares, se determina realizar una corrida para cada metaheurística y cada problema y aumentar considerablemente el tiempo total de ejecución de cada una de las corridas (12 horas).

En cada iteración se obtiene el tiempo de ejecución hasta el momento (entre 0' y 720'), el costo de la mejor solución obtenida en esa iteración y el costo de la mejor solución obtenida hasta ese momento. Además para cada minuto se conoce la descomposición de los diferentes sumandos (separados en función de las restricciones y preferencias) del costo de la mejor solución obtenida hasta ese momento. De esta manera para cualquier minuto en la ejecución se conoce la mejor solución de la iteración actual, la mejor solución global y a su vez la composición de la misma.

Luego se realiza el mismo procedimiento para los problemas 20032-5s/P2 y 20041-5s/P2.

5.4. Valores de los Parámetros

Los parámetros que se utilizan para cada metaheurística se muestran en los cuadros 5.1, 5.2, 5.3, 5.4 y 5.5. Se utilizan los mismos valores de los parámetros para todos los problemas de modo que no se hace diferencia entre un problema y otro.

Parámetro	Valor
Tipo de Población Inicial	C
Tipo de Reproducción	R
Tamaño de la Población	20
Puntos de Cruzamiento	50
Probabilidad de Cruzamiento	0.9
Probabilidad de Mutación	0.005
Probabilidad de Torneo	0.0

Cuadro 5.1: Parámetros utilizados con Algoritmos Genéticos.

Parámetro	Valor
Tipo de Solución Inicial	C
Cantidad de Soluciones Vecinas	500
Largo Lista Tabú Clase	100
Largo Lista Tabú Salón Horario	200

Cuadro 5.2: Parámetros utilizados con Búsqueda Tabú.

Parámetro	Valor
Cantidad de Hormigas	40
Rastro Inicial	50
Evaporación	0.0005
Ponderación del Rastro	10
Ponderación de la Visibilidad	20

Cuadro 5.3: Parámetros utilizados con Ant System.

Parámetro	Valor
Cantidad de Hormigas	10
Rastro Inicial	50
Evaporación	0.05
Evaporación Local	0.1
Probabilidad q0	0.9
Ponderación de la Visibilidad	30

Cuadro 5.4: Parámetros utilizados con Ant Colony System.

Parámetro	Valor
Cantidad de Hormigas	10
Rastro Mínimo	0.2
Rastro Máximo	10
Evaporación	0.001
Ponderación del Rastro	0.5
Ponderación de la Visibilidad	20

Cuadro 5.5: Parámetros utilizados con $\mathcal{MAX} - \mathcal{MIN}$ Ant System.

5.5. Problemas 20032-5 y 20041-5

Los mejores costos obtenidos por las metaheurísticas para los problemas 20032-5 y 20041-5 se muestran en el cuadro 5.6 y también se muestran las diferencias en porcentaje con la mejor solución obtenida.

Metaheurística	Problemas 20032-5		20041-5	
GA	226255	+15.76 %	52737	+104.54 %
TS	195449	0.00 %	38098	+47.76 %
AS	225237	+15.24 %	37163	+44.14 %
ACS	227431	+16.36 %	25783	0.00 %
$\mathcal{MM}$AS	223346	+14.27 %	37821	+46.69 %

Cuadro 5.6: Mejores Costos obtenidos.

5.5.1. Resultados Obtenidos con el Problema 20032-5

Para el problema 20032-5 el mejor resultado se obtiene utilizando Tabú Search y para las otras metaheurísticas se obtienen costos superiores entre un 14.27 % y 16.36 %.

En la figura 5.1 se compara el costo de la mejor solución global obtenida para cada metaheurística en función del tiempo de ejecución y en la figura 5.2 se muestran las diferencias en porcentaje de los costos obtenidos con AS, ACS, $\mathcal{MM}$AS y GA contra el costo obtenido por TS en función del tiempo, donde $costo\_relativo_x = 100.\frac{costo_x - costo_{TS}}{costo_{TS}} \ \forall \ x = \{GA, AS, ACS, \mathcal{MM}AS\}$.

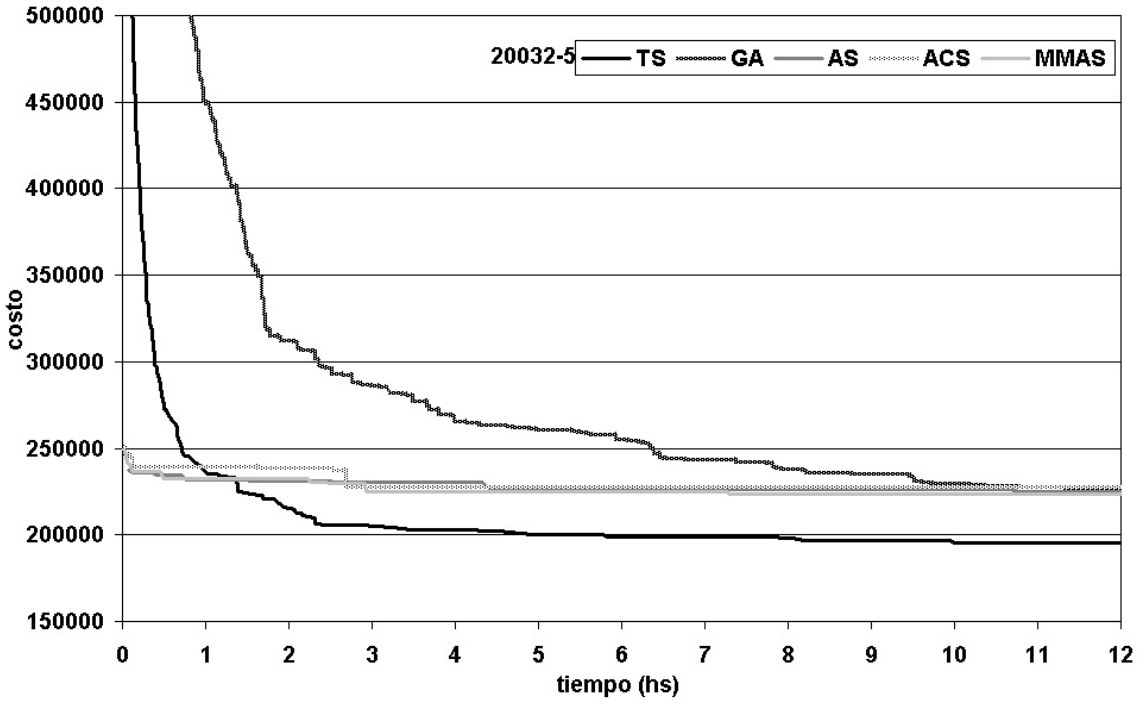


Figura 5.1: Comparación de los Costos obtenidos para 20032-5.

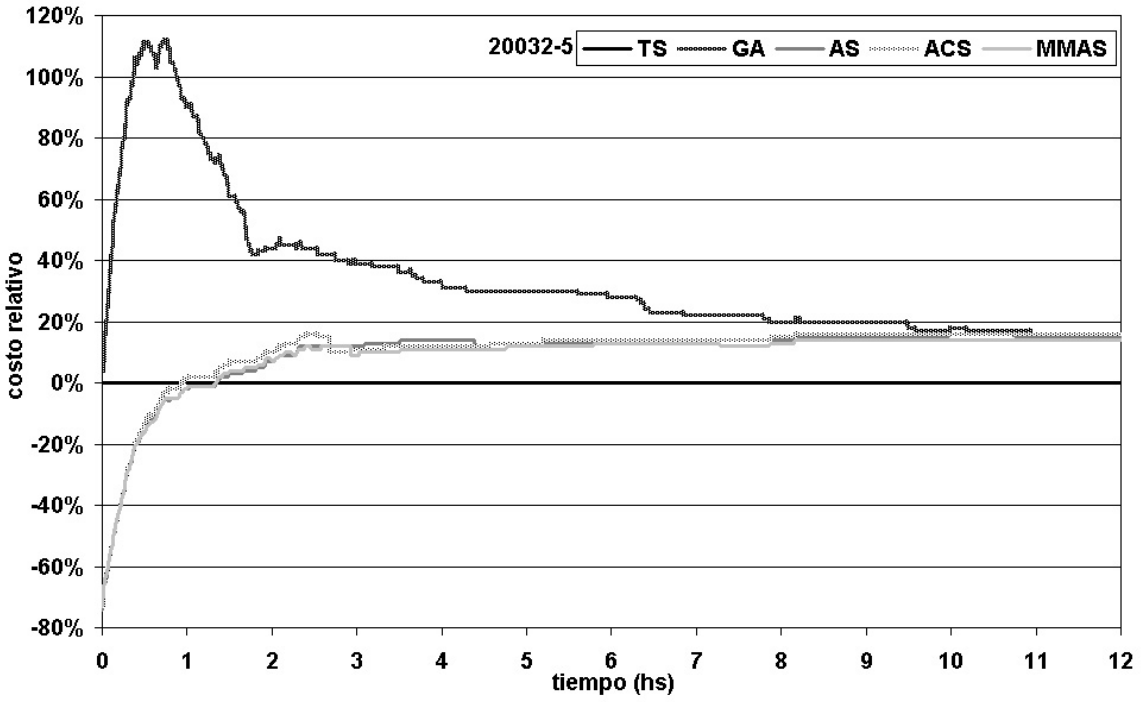


Figura 5.2: Comparación relativa de los Costos obtenidos para 20032-5.

5.5.2. Resultados Obtenidos con el Problema 20041-5

Para el problema 20041-5 el mejor resultado se obtiene utilizando Ant Colony System y para las otras metaheurísticas se obtienen costos entre un 44.14 % y 104.54 % superiores.

En la figura 5.3 se compara el costo de la mejor solución global obtenida para cada metaheurística en función del tiempo de ejecución y en la figura 5.4 se muestran las diferencias en porcentaje de los costos obtenidos con AS, $\mathcal{MM}$AS, TS y GA contra el costo obtenido por ACS en función del tiempo, donde $costo\_relativo_x = 100.\frac{costo_x - costo_{ACS}}{costo_{ACS}} \ \forall \ x = \{TS, GA, AS, \mathcal{MM}AS\}$.

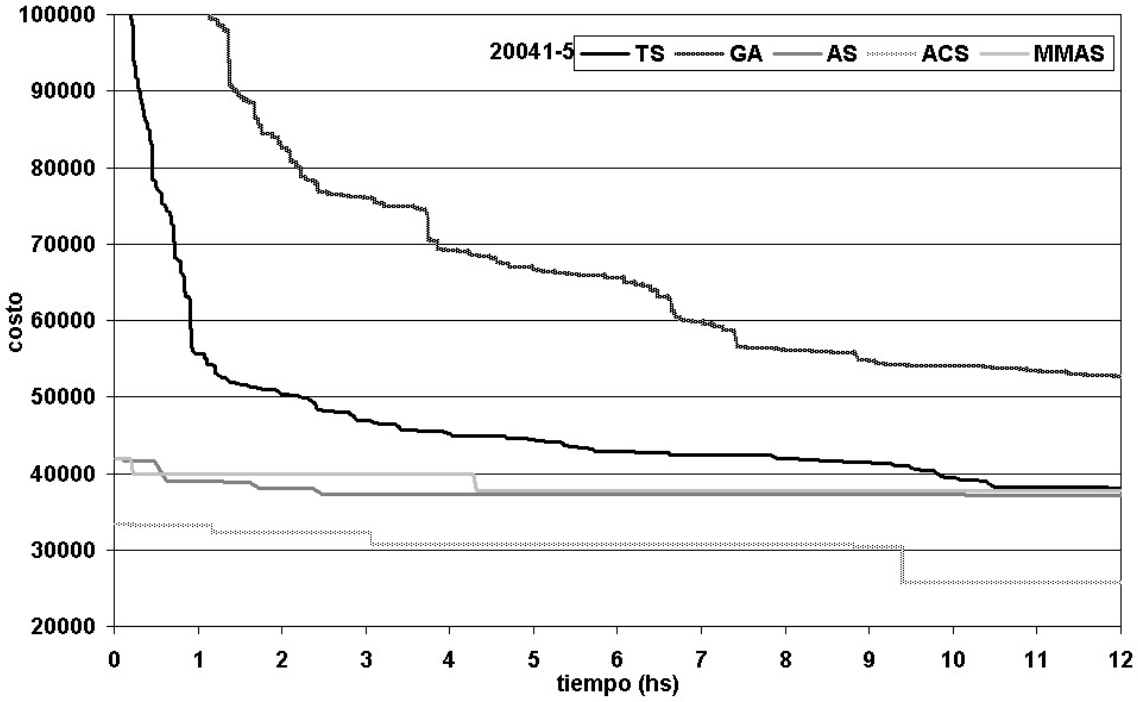


Figura 5.3: Comparación de los Costos obtenidos para 20041-5.

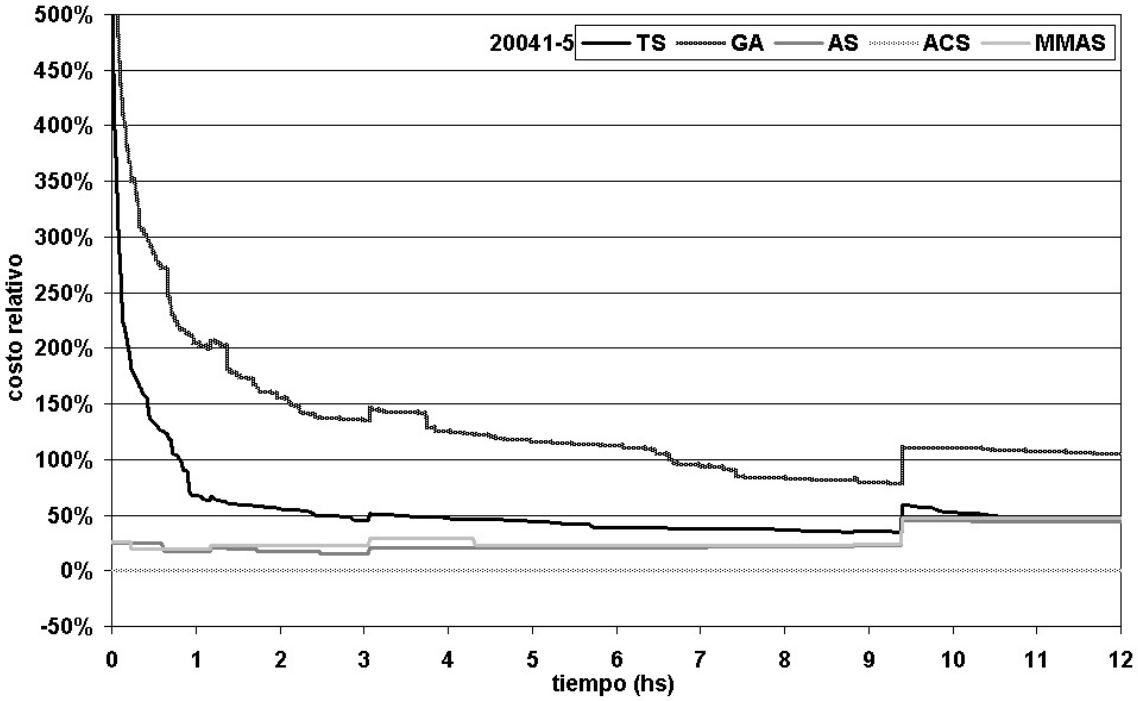


Figura 5.4: Comparación relativa de los Costos obtenidos para 20041-5.

5.5.3. Mejores Soluciones Obtenidas

En las figuras 5.5 y 5.6 se muestran para las mejores soluciones obtenidas por cada metaheurística la descomposición del costo en el correspondiente valor de cada uan de las preferencias y en las figuras 5.7 y 5.8 la descomposición en porcentajes de cada costo. Es de destacar que todas las metaheurísticas utilizadas construyen soluciones factibles, es decir que cumplen con todas las restricciones del problema.

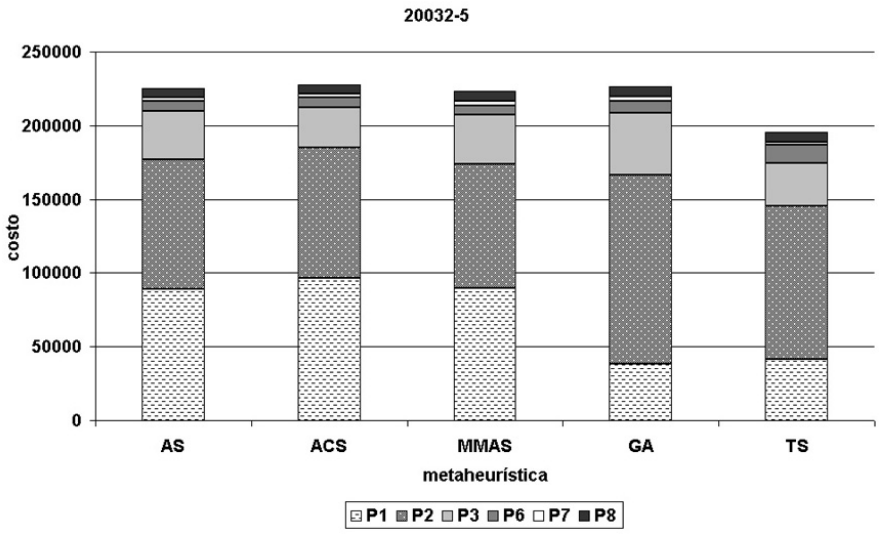


Figura 5.5: Composición del Costo obtenido para 20032-5.

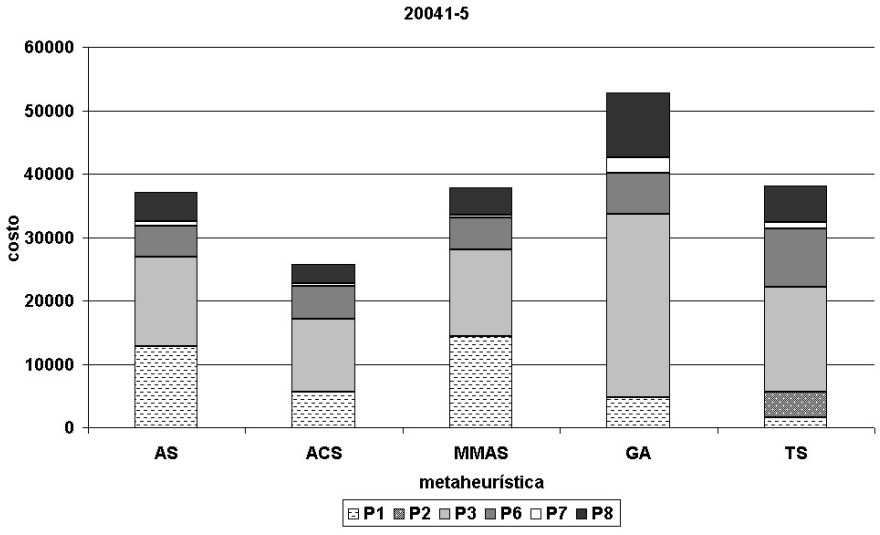


Figura 5.6: Composición del Costo obtenido para 20041-5.

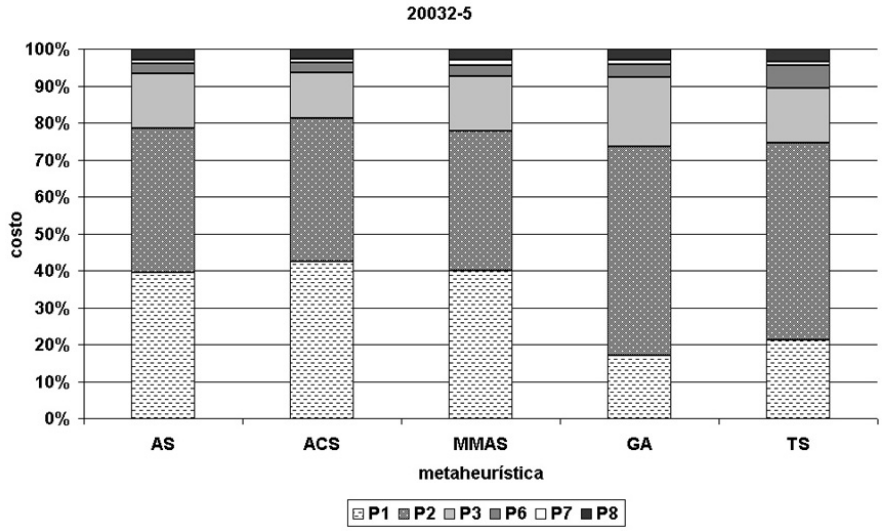


Figura 5.7: Composición en porcentaje del Costo obtenido para 20032-5.

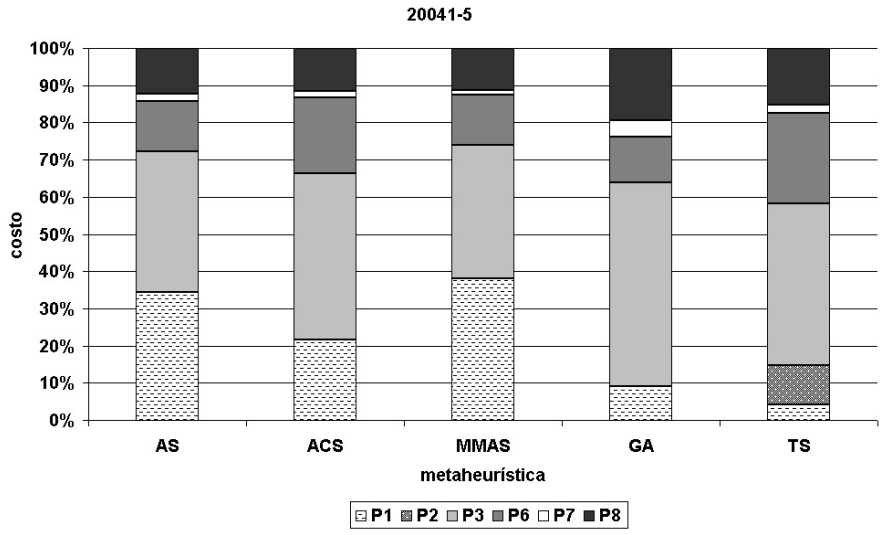


Figura 5.8: Composición en porcentaje del Costo obtenido para 20041-5.

5.6. Problemas 20032-5s/P2 y 20041-5s/P2

Los mejores costos obtenidos para los problemas 20032-5s/P2 y 20041-5s/P2 (los problemas originales 20032-5 y 20041-5 pero sin considerar la preferencia 2[b]) y para cada metaheurística se muestran en el cuadro 5.7 y también se muestran las diferencias en porcentaje con la mejor solución obtenida.

[b]Preferencia de Horario: Todas las clases se programan dentro del horario de preferencia especificado.

	Problemas			
Metaheurística	20032-5s/P2		20041-5s/P2	
GA	39508	+33.93 %	51265	+90.97 %
TS	35481	+20.27 %	34405	+28.17 %
AS	32331	+9.60 %	33666	+25.41 %
ACS	29500	0.00 %	26844	0.00 %
$\mathcal{MM}$AS	33136	+12.33 %	32918	+22.63 %

Cuadro 5.7: Mejores Costos obtenidos 20032-5s/P2 vs. 20041-5s/P2.

5.6.1. Resultados Obtenidos con el Problema 20032-5s/P2

Para el problema 20032-5s/P2 el mejor resultado se obtiene utilizando Ant Colony System y para las otras metaheurísticas se obtienen costos entre un 9.60 % y 33.93 % superiores.

En la figura 5.9 se compara el costo de la mejor solución global obtenida para cada metaheurística en función del tiempo de ejecución y en la figura 5.10 se muestran las diferencias en porcentaje de los costos obtenidos con AS, $\mathcal{MM}$AS, TS y GA contra el costo obtenido por ACS en función del tiempo, donde $costo\_relativo_x = 100.\frac{costo_x - costo_{ACS}}{costo_{ACS}} \ \forall \ x = \{TS, GA, AS, \mathcal{MM}AS\}$.

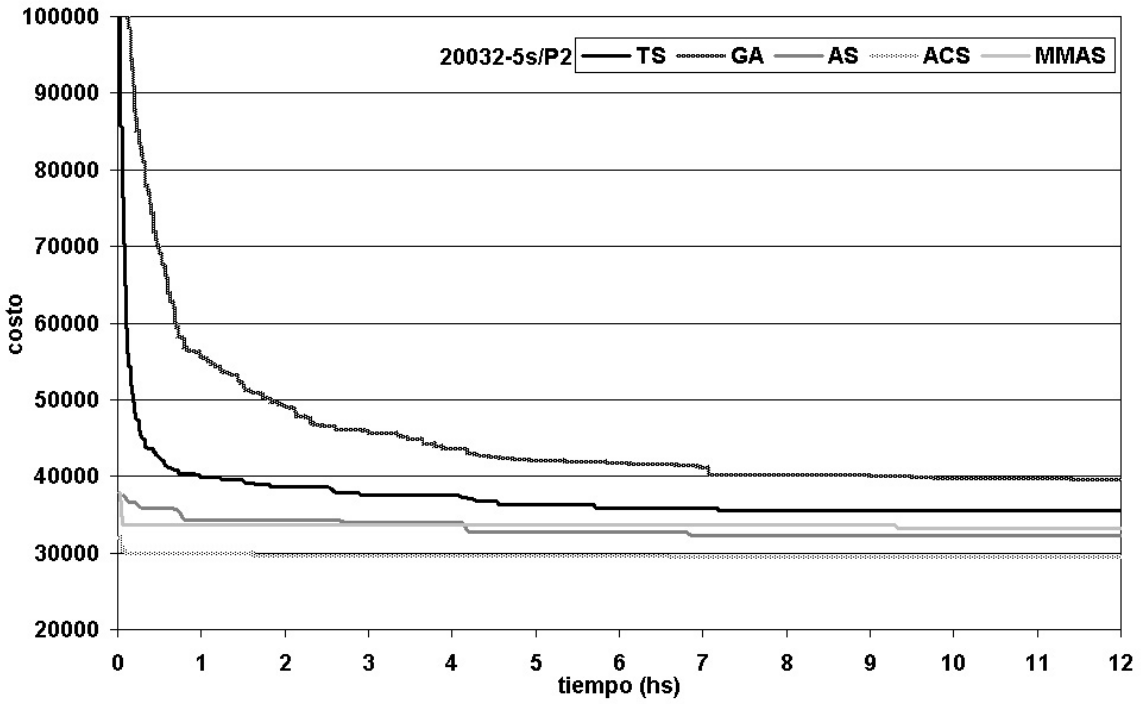


Figura 5.9: Comparación de los Costos obtenidos para 20032-5s/P2.

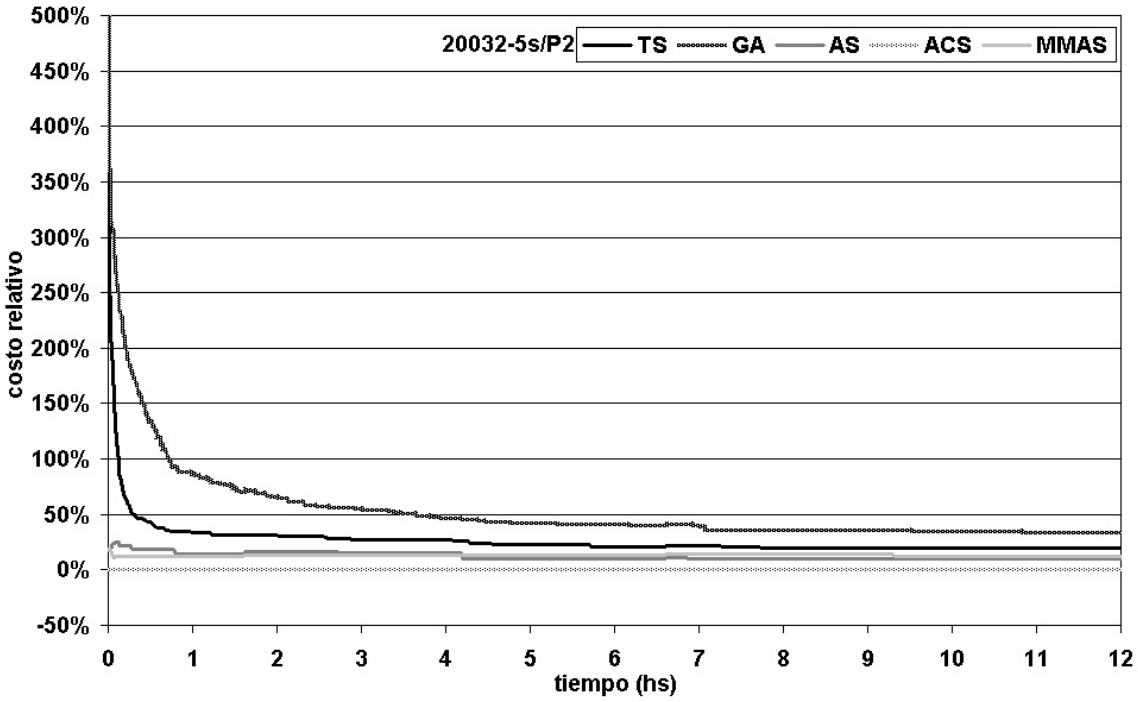


Figura 5.10: Comparación relativa de los Costos obtenidos para 20032-5s/P2.

5.6.2. Resultados Obtenidos con el Problema 20041-5s/P2

Para el problema 20041-5s/P2 el mejor resultado se obtiene utilizando Ant Colony System y para las otras metaheurísticas se obtienen costos entre un 22.63 % y 90.97 % superiores.

En la figura 5.11 se compara el costo de la mejor solución global obtenida para cada metaheurística en función del tiempo de ejecución y en la figura 5.12 se muestran las diferencias en porcentaje de los costos obtenidos con AS, $\mathcal{MM}$AS, TS y GA contra el costo obtenido por ACS en función del tiempo, donde $costo\_relativo_x = 100.\frac{costo_x - costo_{ACS}}{costo_{ACS}} \ \forall \ x = \{TS, GA, AS, \mathcal{MM}AS\}$.

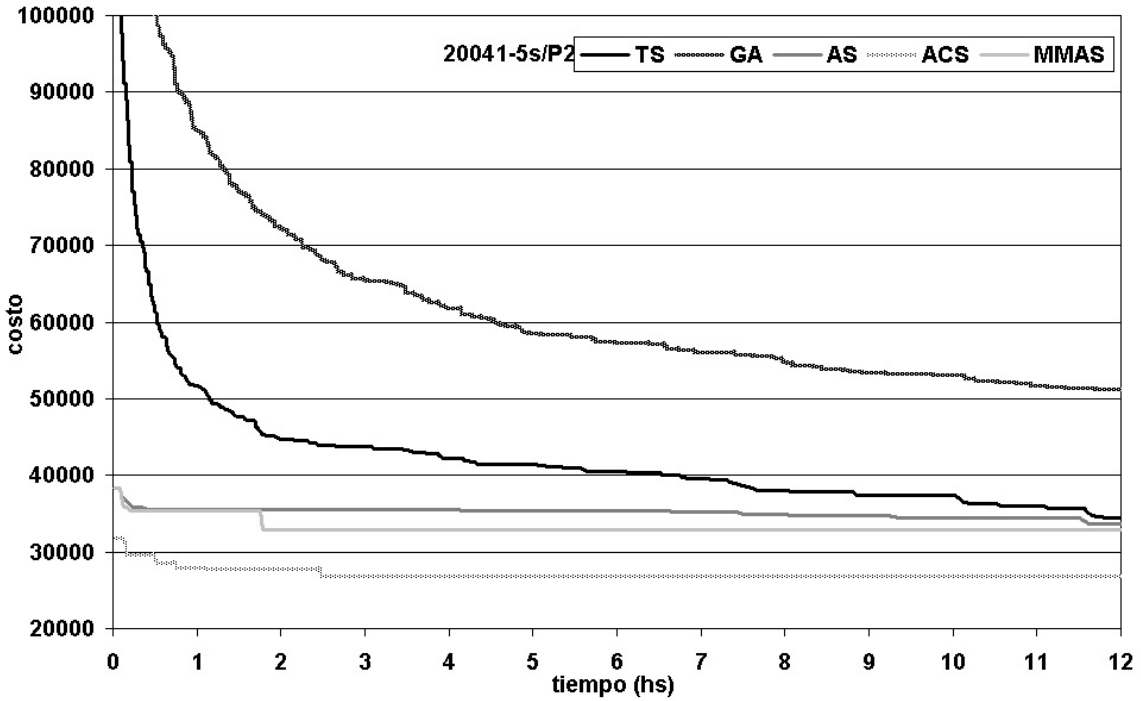


Figura 5.11: Comparación de los Costos obtenidos para 20041-5s/P2.

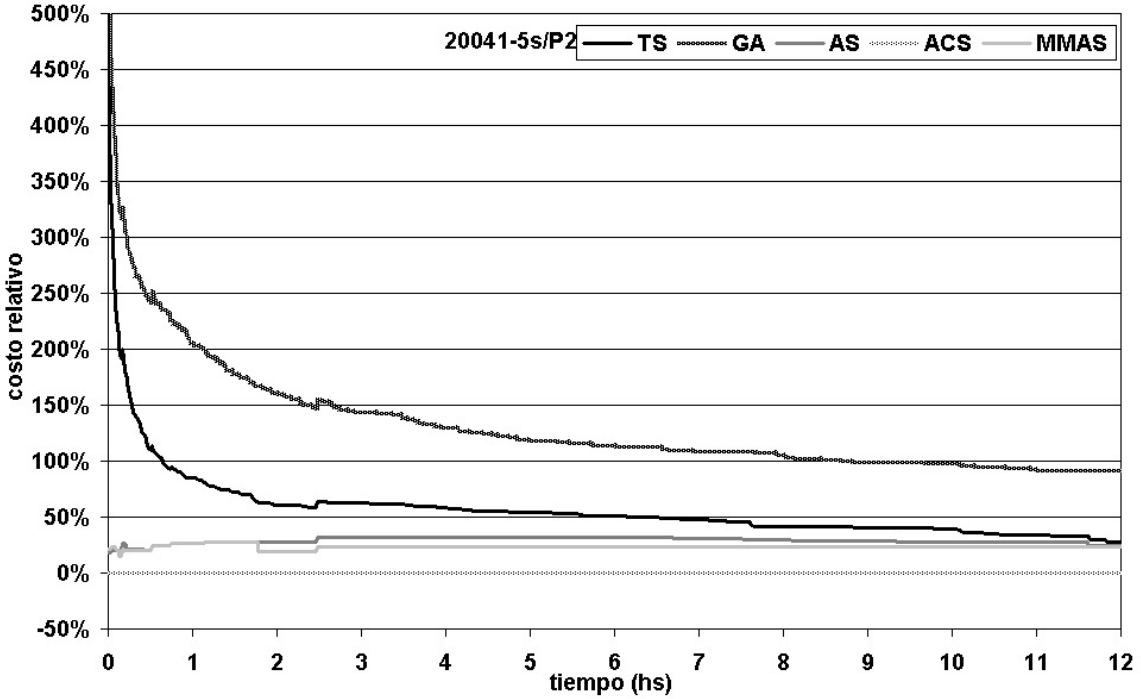


Figura 5.12: Comparación relativa de los Costos obtenidos para 20041-5s/P2.

5.6.3. Mejores Soluciones Obtenidas

En las figuras 5.13 y 5.14 se muestran para las mejores soluciones obtenidas por cada metaheurística la descomposición del costo en el correspondiente valor de cada preferencia y en las figuras 5.15 y 5.16 la descomposición en porcentajes de cada costo. Es de destacar que todas las metaheurísticas utilizadas construyen soluciones factibles, es decir que cumplen con todas las restricciones del problema.

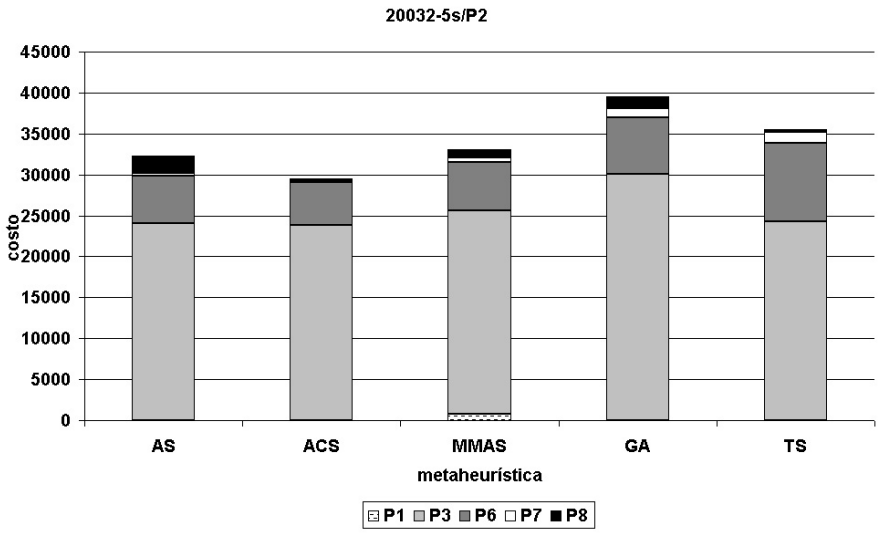


Figura 5.13: Composición del Costo obtenido para 20032-5s/P2.

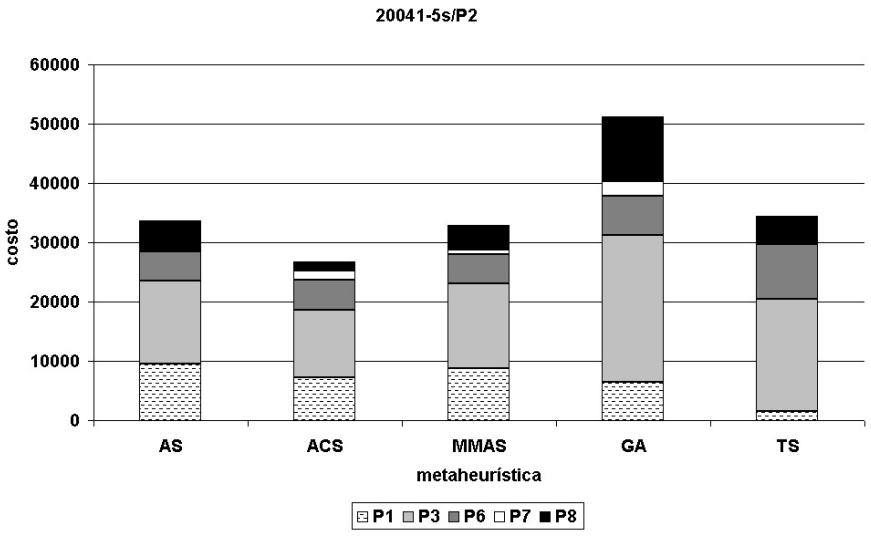


Figura 5.14: Composición del Costo obtenido para 20041-5s/P2.

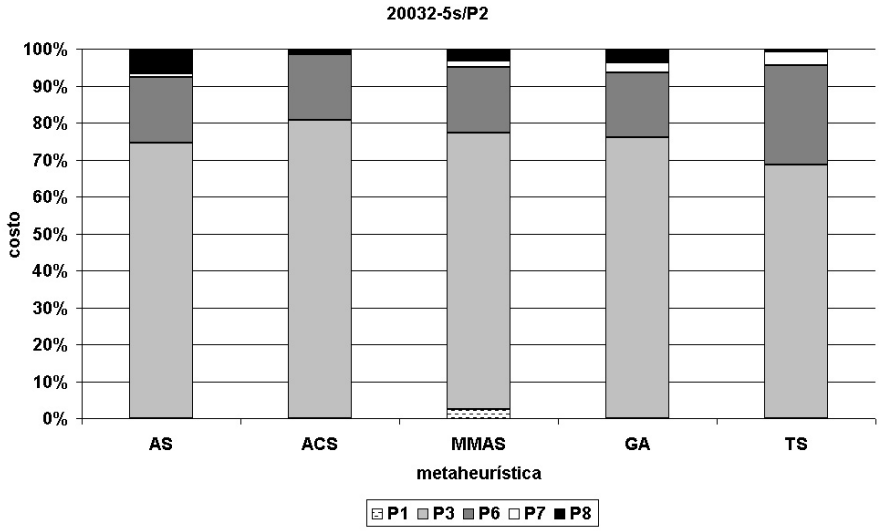


Figura 5.15: Composición en porcentaje del Costo obtenido para 20032-5s/P2.

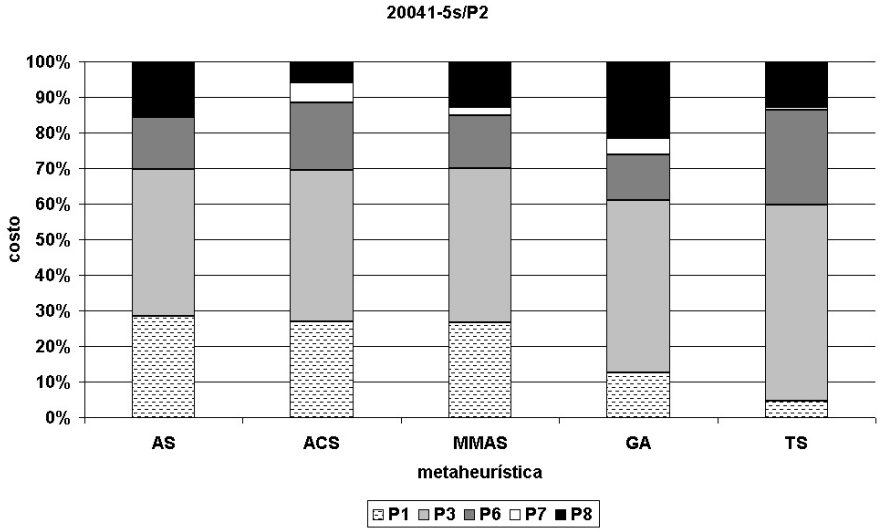


Figura 5.16: Composición en porcentaje del Costo obtenido para 20041-5s/P2.

5.7. Análisis de los Resultados

5.7.1. Problemas 20032-5 y 20041-5

Para el problema 20032-5 el mejor costo se obtiene con TS (195449). Los algoritmos ACO tienen un comportamiento similar, obteniendo costos superiores en el entorno de un $\pm 15\,\%$ (AS-225237-15.24 %, ACS-227431-16,36 % y $\mathcal{MM}$AS-223346-14.27 %) no existiendo diferencias significativas que permitan decir cuál algoritmo es mejor que

otro. Aunque TS es el que da el mejor resultado, recién después de la primer hora logra superar el desempeño de los algoritmos ACO y éstos a partir de esa hora obtienen costos superiores en el entorno de un 15 %. Finalmente GA recién en las últimas dos horas de ejecución logra acercarse a los costos obtenidos con los algoritmos ACO.

Para el problema 20041-5 el mejor costo se obtiene con ACS (25783). Los algoritmos AS y $\mathcal{MM}$AS construyen soluciones de costo superior en el entorno de un ±45 % (AS-37163-44.14 % y $\mathcal{MM}$AS-37821-46.69 %) y durante la ejecución las soluciones obtenidas se mantienen entre un 20 % y 45 % por encima de ACS. En el caso de TS recién en las últimas dos horas de ejecución se aproxima a las soluciones obtenidas con AS y con $\mathcal{MM}$AS. Sin embargo GA no logra converger a las soluciones obtenidas por las otras metaheurísticas obteniendo soluciones de costo superior al doble del obtenido con ACS.

5.7.2. Problemas 20032-5s/P2 y 20041-5s/P2

Para el problema 20032-5s/P2 el mejor costo se obtiene con ACS (29500). Los algoritmos AS y $\mathcal{MM}$AS construyen soluciones de costo superior en el entorno de un ±10 % (AS-32331-9.60 % y $\mathcal{MM}$AS-33136-12.33 %) y durante la ejecución las soluciones obtenidas se mantienen entre un 10 % y 20 % por encima de ACS. En el caso de TS se acerca en costo a menos de un 20 % por encima del costo recién luego de la séptima hora de ejecución y GA se mantiene lejos de los costos obtenidos con ACS durante toda la ejecución.

Para el problema 20041-5s/P2 el mejor costo se obtiene con ACS (26844). Los algoritmos AS y $\mathcal{MM}$AS construyen soluciones de costo superior en el entorno de un ±24 % (AS-33666-25.41 % y $\mathcal{MM}$AS-32918-22.63 %) y durante la ejecución las soluciones obtenidas se mantienen entre un 20 % y 30 % por encima de ACS. En el caso de TS recién en la última hora de ejecución se aproxima a menos de un 30 % por encima del costo obtenido con ACS. Sin embargo GA no logra converger a las soluciones obtenidas por las otras metaheurísticas obteniendo soluciones de costo el doble del obtenido con ACS.

5.7.3. Conclusiones

Las diferentes metaheurísticas no se comportan igual para los dos semestres analizados, para el segundo semestre del 2003 se obtiene la mejor solución con Tabú Search y para el primer semestre del 2004 con Ant Colony System. Esto se debe a que el comportamiento de las diferentes metaheurísticas está fuertemente influenciado por los datos del problema real. El problema para el segundo semestre del 2003 está completo y en el primer semestre del 2004 faltan la mayoría de los datos de una de las

preferencias.

Al redefinir el problema eliminando la preferencia incompleta se obtienen resultados similares para los dos semestres, en ambos Ant Colony System es el que obtiene la mejor solución, seguidos por los otros dos algoritmos ACO y luego por Búsqueda Tabú y por último por Algoritmos Genéticos.

Algoritmos ACO

El comportamiento de los algoritmos ACO es similar independiente del problema, en los primeros minutos comienzan construyendo soluciones muy buenas y no logran grandes mejoras a pesar de estar varias horas ejecutándose, por lo que no se logra gran convergencia de los algoritmos.

Puntualmente, el comportamiento de AS y $\mathcal{MM}$AS es similar independientemente del problema analizado, tanto en velocidad de convergencia como en las soluciones obtenidas. Sin embargo, en el caso de ACS se tiene el mismo comportamiento solo para el problema 20032-5 (problema con los datos completos), pero obtiene mejores soluciones para los otros problemas.

Se puede concluir que dentro de los algoritmos ACO, ACS parece ser la mejor opción para los problemas sin la preferencia 2 y que para el problema completo, se obtienen resultados similares con las tres opciones.

Búsqueda Tabú

Con Búsqueda Tabú se obtiene el mejor resultado para el problema completo (20032-5), pero para los otros problemas recién en las últimas horas de ejecución logra aproximarse a las soluciones obtenidas con ACO.

Aunque Búsqueda Tabú en un problema obtiene la mejor solución y para los otros problemas recién en las últimas horas de ejecución se acerca a las soluciones obtenidas por los algoritmos ACO, en la primera hora de ejecución siempre los mejores resultados se obtienen con los algoritmos ACO.

Algoritmos Genéticos

En todos los casos analizados los Algoritmos Genéticos son los que obtienen la peor solución.

5.7.4. Búsqueda Tabú vs. Ant Colony System

Para el problema completo (20032-5) se obtiene la mejor solución con Búsqueda Tabú, pero para los otros problemas considerados ACS es con el que se obtiene la mejor solución.

La preferencia 2 para el problema 20041-5 casi que no se considera, esto es por los datos cargados del problema y a diferencia del problema 20032-5 que si se indica para todas las clases el horario de preferencia y el turno para el grupo. Esto hace que las metaheurísticas para el problema 20032-5 se encuentren muy limitadas en encontrar mejores soluciones por el peso de esa preferencia en el costo total (entre un 40 % y 60 %). Esto podría hacer que las diferentes metaheurísticas lleguen a un nivel de soluciones que no puedan mejorar, ya que una mejora en los costos de las otras preferencias, hace que varíe muy poco la visibilidad o la aptitud. Lo que hace que probabilísticamente sea similar elegir esas soluciones. Sin embargo Tabú Search no utiliza esa información relativa al costo por lo que se puede justificar el porque obtiene la mejor solución para este problema.

Para el problema 20041-5 al no tener el peso en el costo de la preferencia 2, los cambios en las otras preferencias afectan en mayor medida la visibilidad y la aptitud, por lo que en este caso las metaheurísticas se comportan diferente para este problema. Similar es el caso para los problemas modificados 20032-5s/P2 y 20041-5s/P2.

Capítulo 6

Conclusiones

En este último capítulo se presentan los principales objetivos alcanzados en este trabajo, así como las conclusiones a las que se puede llegar luego del desarrollo de todo el trabajo. Algunas especulaciones sobre los resultados obtenidos y finalmente las posibles direcciones para encarar una investigación futura también son presentados.

6.1. Objetivos Alcanzados

Esta tesis se planteó como objetivo principal la adaptación, implementación y aplicación de un conjunto de metaheurísticas para el Problema de Asignación de Salones y Horarios a Asignaturas.

Los principales objetivos planteados fueron:

- Realizar una actualización del estado del arte en cuanto al problema de programación de horarios.
- Construir un motor de metaheurísticas.
- Obtener un informe sobre calibrado y comparación de las mismas.

Como primera etapa del trabajo, se realizó un exhaustivo estudio sobre los diferentes métodos existentes utilizados en la resolución de los diferentes tipos de problema de programación de horarios. Para esto primero se presentó formalmente el problema, sus diferentes variaciones y un estudio de los diferentes métodos y técnicas utilizados para su solución. Dentro de esos métodos y técnicas se nombran: métodos exactos, métodos secuenciales, métodos de agrupamiento, enfoque basado en restricciones, métodos metaheurísticos y métodos hiperheurísticos. Como resultado se obtiene una extensa lista de referencias bibliográficas.

De ese estudio solo se encontró una investigación que compare diferentes metaheurísticas aplicadas a un problema de programación de horarios [RDSC+02] de forma similar al realizado en esta tesis, enmarcado dentro de las actividades de investigación del proyecto Europeo "Metaheuristic Network"[a].

Como segunda etapa de este trabajo, se describió y se formalizó el problema real de la Facultad (PASHA) y luego se presentó su formulación matemática. Luego se realizó un estudio de las características principales de las metaheurísticas más utilizadas (Algoritmos Genéticos, Búsqueda Tabú, Ant System, Ant Colony System y $\mathcal{MAX} - \mathcal{MIN}$ Ant System) y su adaptación e implementación al problema real de la Facultad.

Luego de implementadas, se realizó una calibración de los diferentes parámetros que controlan los algoritmos y finalmente la obtención de los resultados aplicando las mismas a instancias reales del problema.

6.2. Conclusiones

Las principales conclusiones obtenidas en este trabajo son:

Desempeño de las Metaheurísticas

El buen o mal desempeño de las diferentes metaheurísticas implementadas depende fuertemente de los datos del problema. Para el problema con todos los datos completos (20032-5) el mejor resultado se obtiene con Búsqueda Tabú, para las otras metaheurísticas, al cabo de un tiempo prolongado de búsqueda, se obtienen resultados similares, en el entorno de un 15 % superiores. Sin embargo, para el problema sin considerar la preferencia 2[b] los mejores resultados se obtienen con Ant Colony System, seguido de Ant System, $\mathcal{MAX} - \mathcal{MIN}$ Ant System y finalmente Búsqueda Tabú y Algoritmos Genéticos.

Los diferentes algoritmos ACO implementados y aplicados a los diferentes problemas considerados tiene un comportamiento similar a lo largo de la ejecución, en velocidad para encontrar buenas soluciones y en la calidad de las mismas.

En todas las instancias del problema estudiado, Algoritmos Genéticos es con el que se obtiene los peores resultados. Un justificativo se puede encontrar en que la gran mayoría de los cruzamientos generan soluciones no factibles que son descartadas. En la sección 6.4.1 se encuentran algunas sugerencias para intentar mejorar la convergencia.

[a] Metaheuristic Network - http://www.metaheuristics.org

[b] Preferencia de Horario: Todas las clases se programan dentro del horario de preferencia especificado.

Con los resultados obtenidos, en principio, considerando el problema completo, como se definió originalmente y con todos los datos, se debería considerar a Búsqueda Tabú como la mejor opción. Pero sin duda, al ser un problema cuya solución depende fuertemente de los datos, a priori para cada semestre nuevo considerado para realizar la asignación, en principio, se debería tener en cuenta al menos a Búsqueda Tabú y Ant Colony System como métodos para solucionarlo.

Calibración de los Parámetros

Las metaheurísticas no presentan gran sensibilidad a los cambios de los valores de los parámetros, de la calibración se deduce que pudiendo definir valores razonables para c/u de ellos, teniendo en cuenta consideraciones de tiempo de ejecución, memoria, etc. las metaheurísticas convergen a soluciones razonables y no muy diferentes en término de costo entre sí. Por eso no se utilizan otras técnicas existentes, como ser análisis de factores [LK91] y si solamente ensayo y error.

Soluciones iniciales aleatorias vs. construidas

Tanto Algoritmos Genéticos como Búsqueda Tabú necesitan una o mas soluciones iniciales, la elección de soluciones aleatorias parece generar diversidad pero su costo esta muy lejos de las soluciones aceptables, lo que hace que la convergencia sea muy lenta y no se hace manejable en tiempos razonables. Por eso con los tiempos manejados en este trabajo (máximo de 12 horas), tanto para TS como para GA se obtienen mejores resultados partiendo de soluciones construidas y no aleatorias. Igual queda abierta la posibilidad de investigar el comportamiento de estas metaheurísticas con soluciones iniciales aleatorias (siempre y cuando se logre mejorar la velocidad de convergencia de ambos métodos). En todos los casos las soluciones iniciales construidas por el algoritmo 19 son de mejor costo que las soluciones manuales construidas en bedelía y las obtenidas en el proyecto de grado [CPU03].

6.3. Evolución del Modelo

Originalmente, en el proyecto de grado [CPU03] el problema se formuló considerando como restricción la superposición de clases[c] y como preferencia el que todas las clases sean programadas. De esta manera se le daba mas importancia a que no se superpusieran las clases de una misma asignatura con un solo grupo a que todas las clases sean programadas.

[c]Superposición: Las clases de una asignatura básica u obligatoria con un solo grupo no se superponen.

Considerando el problema original, debido a los datos reales, los algoritmos ACO son los únicos que encuentran soluciones factibles, y Algoritmos Genéticos y Búsqueda Tabú construyen soluciones no factibles y no logran converger a soluciones factibles. De modo que, con ésta situación, no es posible realizar una comparación real de las metaheurísticas y se decide modificar la definición del problema para permitir que todas las metaheurísticas consideradas converjan a soluciones factibles. Para eso se considera la superposición de clases como una preferencia y que todas las clases sean programadas como una restricción (restricción 5 y preferencia 1).

6.4. Investigaciones y Trabajo Futuro

De los resultados obtenidos en el presente trabajo se dejan planteadas algunas ideas para realizar en un futuro:

- Extender la investigación a los otros tipos de problemas de programación de horarios que tiene la Facultad:
  - Programación de Horarios de Clases incorporando los profesores, ampliando el problema para incorporar en la asignación a los profesores.
  - Programación de Horarios de Parciales y Exámenes, en el cual en vez de considerar una semana en el cual asignar las clases regulares, tomar el tiempo total del período y asignarles los diferentes parciales y exámenes.
- Realizar mejoras a las metaheurísticas implementadas, ver sección 6.4.1.
- Implementar la preferencia 5[d] que en el proyecto de Taller V original [CPU03] no se implementó por su complejidad y en este trabajo tampoco se tuvo en cuenta.
- Incorporar nuevas restricciones y preferencias, es razonable asumir que desde el año 2003 hasta la fecha en la Bedelía de la Facultad cuentan con experiencia en el uso de la aplicación y las condiciones planteadas en el año que se hizo el relevamiento del problema hayan variado.
- Incorporar las metaheurísticas implementadas al sistema que utilizan en bedelía, de manera de brindarle al usuario la opción de elegir que método utilizar y así también poder utilizar la solución obtenida por uno de los algoritmos (por ejemplo de ACS) como solución inicial para TS, etc.
- Realizar un estudio en el resto de la Universidad de la República analizando la posibilidad de extender el uso de la herramienta a las otras facultades.

[d] Contigüidad Horaria: Los teóricos y prácticos de una asignatura se dictan en horarios contiguos.

- Incorporar otras metaheurísticas, como Simulated Annealing, Algoritmos Meméticos e investigar la aplicación de métodos híbridos.

6.4.1. Mejoras de las Metaheurísticas

Se implementaron las diferentes metaheurísticas adaptando las mismas en su forma básica. Algunas ideas que se pueden incorporar a las metaheurísticas para tratar de mejorar la performance de cada una de ellas son dadas a continuación.

Algoritmos ACO

- Incorporar la búsqueda local para ACS y $\mathcal{MM}$AS, que por razones de eficiencia en tiempo no se tuvo en cuenta. Analizar si la demora al realizar una búsqueda local se justifica en función de la obtención de mejores soluciones.
- Incorporar a los algoritmos las adaptaciones realizadas en uno de los proyectos anteriores [RC99]. Una de ellas es la de realizar asignaciones en bloque, una vez que se asigna la primera clase de un grupo asignar las restantes al mismo salón y horario en los otros días de la semana. De esta manera el algoritmo se hace mas "determinístico", pero se debe evaluar si es conveniente o no según el resto de las restricciones y preferencias.
- Realizar un estudio mas detallado del efecto del rastro y de la visibilidad en la regla de transición.
- Variar la cantidad de rastro depositado en relación al costo de la solución obtenida. Cambiar $1/C(S)$ por $k/C(S)$ incorporando un nuevo parámetro que determine la cantidad de feromona depositada en cada arista en función del costo.
- Recorrer los salones x horario asignando las clases [RC99].
- Debido a que mientras una hormiga construye una solución, para una clase puede existir un conjunto muy grande de posibles movimientos (salón x horario) incrementando el tiempo de ejecución. En estas situaciones, se puede limitar el tiempo utilizando una lista de movimientos candidatos conteniendo los movimientos mas promisorios del vecindario.

Algoritmos Genéticos

- Cambiar la Función de Aptitud en función del valor de la Función Objetivo del problema. La aptitud se calcula como $1/C(S)$, pero se puede variar a $1/(1 + C(S))$ o $1/(1 + C(S)^2)$ o también escalar la aptitud antes de la selección.

- Variar las probabilidades de cruzamiento y mutación durante la ejecución
- Dejar el mejor padre de una generación a otra, o sea no sustituir toda la población, sino hacer sustitución parcial (elitismo).
- Aplicar búsqueda local en cada cromosoma.
- Mejorar las operaciones de Cruzamiento y Mutación (Check and Repair) de manera que de su aplicación se obtengan cromosomas factibles, es decir, que no transformen soluciones factibles en soluciones no factibles y que haga que se descarten de la población
- Implementar el cruzamiento uniforme (Uniform Crossover).
- Para cada cromosoma identificar las clases asignadas que aportan menor aptitud y aplicarles la operación de Mutación (Violated Direct Mutation).

Apéndice A

Metaheurísticas

A.1. Introducción

Las Metaheurísticas [CDM+96] surgen derivadas de la observación de los procesos que ocurren en los fenómenos físicos, biológicos y sociales de la naturaleza, extrayendo de esos fenómenos ciertas características que pueden ser aplicadas en la resolución de los problemas de optimización combinatoria $\mathcal{NP}$-completos. Una de las características que tienen las metaheurísticas es que han sido definidas de manera de que sean robustas (aplicables a una gran variedad de problemas con una mínima, sí fuera necesario, modificación de su definición básica) y efectivas como herramientas de optimización, las cuales comienzan con una solución inicial o una población de individuos iniciales y se van modificando iterativamente hasta que se llegue a la condición de fin, la cual se puede definir como haber obtenido una solución considerada aceptable o bien llegar al número máximo de iteraciones.

Cada metaheurística cumple que:

- Se aplica utilizando cierto número de ciclos, hasta satisfacer criterio de fin.
- Utiliza uno o más "agentes" (neuronas, partículas, cromosomas, hormigas, etc.).
- En caso de múltiples agentes tiene definido un mecanismo de competición-cooperación.
- Permite modificar los parámetros y la representación del problema.

En este capítulo se presentan una descripción detallada de cada metaheurística implementada en esta tesis: Algoritmos Genéticos, Búsqueda Tabú, Ant System, Ant Colony System y $\mathcal{MAX} - \mathcal{MIN}$ Ant System.

A.2. Algoritmos Genéticos

A.2.1. Introducción General

Los Algoritmos Genéticos (Genetic Algorithms, GA) fueron introducidos primero por Holland [Hol75] como un algoritmo robusto de búsqueda. Mas tarde, especialmente en los trabajos de Goldberg [Gol89] fueron utilizados como técnica de optimización.

Los GA son métodos adaptativos que se pueden utilizar para implementar búsquedas y problemas de optimización. Ellos están basados en los procesos genéticos de organismos biológicos, codificando una posible solución al problema en un cromosoma compuesto por una cadena de bits o caracteres. Estos cromosomas representan individuos que son llevados a lo largo de varias generaciones, en forma similar a las poblaciones naturales, evolucionando de acuerdo a los principios de selección natural y supervivencia del más apto, descritos por primera vez por Charles Darwin en su libro "Origen de las Especies".

Emulando los procesos de selección natural y supervivencia del más apto, los GA son capaces de evolucionar soluciones a problemas del mundo real. En la naturaleza, los individuos compiten entre sí por recursos tales como comida, agua y refugio. Adicionalmente, los animales de la misma especie normalmente antagonizan para obtener una pareja. Aquellos individuos que tengan más éxito tendrán probablemente un número mayor de descendientes, por lo tanto, mayores probabilidades de que sus genes (unidad básica de codificación de cada uno de los atributos de un ser vivo) sean propagados a lo largo de sucesivas generaciones. La combinación de características de los padres bien adaptados en un descendiente, puede producir muchas veces un nuevo individuo mejor adaptado que cualquiera de sus padres a las características del medio ambiente. Holland, a fines de la década del 60, estaba consciente de la importancia de la selección natural y desarrolló una técnica que permitió incorporarla en un programa de computadora. Y los objetivos de su investigación fueron dos:

- abstraer y explicar rigurosamente el proceso adaptativo de los sistemas naturales y
- diseñar sistemas artificiales que retuvieran los mecanismos más importantes de los sistemas naturales.

Resumiendo, su objetivo era lograr que las computadoras aprendieran por sí mismas. A la técnica que inventó Holland se le llamó originalmente Planes Reproductivos, pero se hizo popular bajo el nombre de Algoritmos Genéticos tras la publicación de su libro [Hol75].

En este sentido los GA son: *"Algoritmos de búsqueda basados en los mecanismos de selección natural y genética natural. Combinan la supervivencia de los más compatibles entre las estructuras de cadenas, con una estructura de información ya aleatorizada, intercambiada para construir un algoritmo de búsqueda con algunas de las capacidades de innovación de la búsqueda humana."*

Una definición bastante completa es la dada por John Koza [Koz92]: *"Es un algoritmo matemático altamente paralelo que transforma un conjunto de objetos matemáticos individuales con respecto al tiempo usando operaciones modeladas de acuerdo al principio Darwiniano de reproducción y supervivencia del más apto y tras haberse presentado de forma natural una serie de operaciones genéticas de entre las que se destaca la recombinación sexual. Cada uno de estos objetos matemáticos suele ser una cadena de caracteres (letras o números) de longitud fija que se ajusta al modelo de las cadenas de cromosomas y se les asocia con una cierta función matemática que refleje su aptitud."*

Los GA utilizan una analogía directa del fenómeno de la evolución en la naturaleza. Trabajan con una población de individuos, cada uno representando una posible solución a un problema dado. A los individuos más adaptados se les da la oportunidad de reproducirse mediante cruzamientos con otros individuos de la población, produciendo descendientes con características de ambos padres. Los miembros menos adaptados poseen pocas probabilidades de que sean seleccionados para la reproducción y desaparecen. Una nueva población de posibles soluciones es generada mediante la selección de los mejores individuos de la generación actual, emparejándolos entre ellos para producir un nuevo conjunto de individuos. Esta nueva generación contiene una proporción más alta de las características poseídas por los mejores miembros de la generación anterior. De esta forma, a lo largo de varias generaciones, las buenas características son difundidas a lo largo de la población mezclándose con otras. Favoreciendo el emparejamiento de los individuos mejor adaptados es posible recorrer las áreas más prometedoras del espacio de búsqueda.

Existen tres operaciones básicas que contribuyen a la evolución:

- Reproducción: cada individuo se reproduce según cierta probabilidad.
- Cruzamiento: dos individuos cruzan su material genético, o sea cada uno es dividido en dos o mas partes y las partes son cruzadas. De esta manera dos nuevos individuos son creados.
- Mutación: es la introducción en forma aleatoria de un nuevo individuo, usualmente producto del cambio de un gen de un individuo. Esta operación no debe ser utilizada demasiado (generalmente en el orden de 1 en 1000) ya que el GA se puede convertir en una búsqueda al azar, pero su utilización asegura que todos los puntos en el espacio de búsqueda tengan probabilidad de ser examinados.

Dado un problema de optimización, se define la Función de Aptitud (Fitness) como la función objetivo del problema y cada individuo de la población representa una de las soluciones factibles. Los GA tienen como objetivo encontrar un individuo de la población que tenga el mejor valor de aptitud. Inicialmente el algoritmo define en el tiempo 0 una población inicial $S_1^0, S_2^0, ..., S_p^0$ de p individuos, generalmente seleccionados de forma aleatoria, para la cual se busca cual es el mejor individuo, donde cada individuo consiste en una colección (usualmente un string) de elementos atómicos llamados "genes". Cada gen toma su valor en un conjunto predeterminado. En cada paso t del algoritmo se realiza la generación $S_1^t, S_2^t, ..., S_p^t$ y se trabaja sobre la población modificando sus componentes. Las modificaciones ocurren de acuerdo a las reglas genéticas, implementadas por las operaciones genéticas definidas anteriormente. La reproducción selecciona uno por uno los individuos en $t+1$ a partir de los de t a través de un sorteo. El sorteo se realiza según las probabilidades de reproducción basadas en su función de aptitud. Luego a cada par de individuos de la población obtenida se le aplica el cruzamiento con una probabilidad de cruzamiento p_c. Y finalmente a la población obtenida se le aplica la mutación con probabilidad de mutación p_m. Cuando el criterio de parada es satisfecho se devuelve la mejor solución obtenida. Generalmente se usan dos criterios de parada: correr el algoritmo durante un número máximo de generaciones o detenerlo cuando la población se haya estabilizado, o sea cuando todos o la mayoría tengan la misma aptitud. Los principales parámetros que controlan el método son:

- el tamaño de la población p,
- la probabilidad de cruzamiento p_c, y
- la probabilidad de mutación p_m.

La estructura genérica del algoritmo GA se muestra en el algoritmo 20.

Algorithm 20 *Metaheurística GA*

```
1: InicializarPoblación(S^0)
2: EvaluarPoblación(S^0)
3: while criterio terminación no satisfecho do
4:    S^{t+1}= Reproducción(S^t)
5:    S^{t+1}= Cruzamiento(S^{t+1})
6:    S^{t+1}= Mutación(S^{t+1})
7:    t = t + 1
8:    EvaluarPoblación(S^t)
9: end while
```

El poder de los GA proviene del hecho de que la técnica es robusta y puede manejar exitosamente un amplio rango de problemas, incluso algunos que son difíciles de resolver por otros métodos. Los GA no garantizan que encontrarán la solución óptima

al problema, pero son generalmente buenos encontrando en corto tiempo buenas soluciones a los problemas. Donde existan técnicas especializadas para la resolución de problemas, éstas superarán fácilmente a los GA tanto en velocidad como en precisión. El campo principal de aplicación es en donde no existan este tipo de técnicas o donde no se puedan aplicar.

A.2.2. Codificación de la Solución

En GA, un cromosoma (que representa una solución al problema) es codificado como un string de símbolos. Existen dos maneras de codificar según la cardinalidad de los símbolos utilizados: la codificación binaria y la codificación con múltiples símbolos. La codificación mas comúnmente usada es la binaria, utilizada inicialmente por Holland en sus primeras investigaciones, es simple y puede representar la mayor información con un mínimo número de bits. Sin embargo, la utilización de codificación con múltiples símbolos puede proveer beneficios que la codificación binaria no puede brindar. Puede ayudar a evitar que se creen cromosomas no válidos en las operaciones genéticas, por ejemplo si una variable puede tomar cinco valores válidos, tres bits son necesarios para representarla y con tres bits los cinco valores válidos pueden representarse por los valores binarios de 0 a 4, valores binarios de 5 a 7 generados por las operaciones no son válidos, figura A.1.

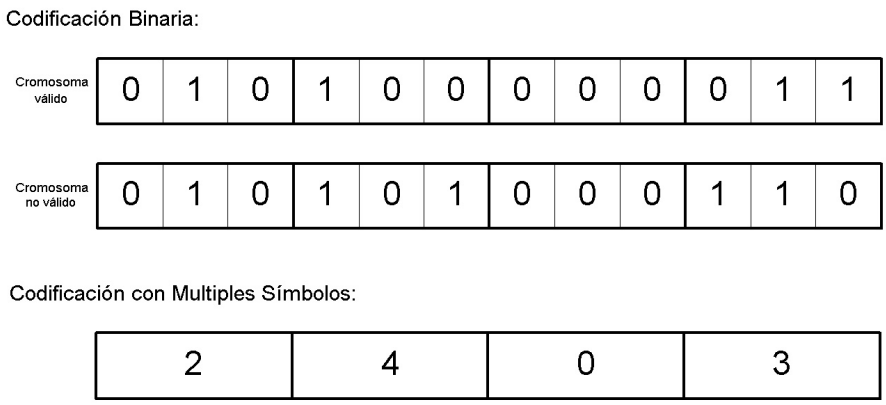


Figura A.1: Codificación de los Cromosomas.

También los cromosomas se pueden representar mediante la codificación directa o la codificación indirecta. La codificación directa es simple, una solución es directamente representada por un cromosoma. Las violaciones a las restricciones son penalizadas directamente en la aptitud del cromosoma. Una función de castigo es incluida en la función de aptitud. La codificación indirecta no siempre es posible y efectiva. Cuando el espacio de soluciones es altamente restrictivo, se necesitan operadores específicos que restrinjan la búsqueda solamente en el espacio de soluciones factibles. Un esquema alternativo es codificar solamente la información bruta de la solución en vez de la solución entera, y un constructor de las soluciones que incluya la información de las

restricciones es usado en el proceso de construcción de soluciones legales. Con este esquema se pueden usar una codificación simple de los cromosomas y los operadores genéticos.

Respecto a la codificación de la solución, uno de los resultados fundamentales de la teoría de los algoritmos genéticos, el teorema de los esquemas, afirma que la codificación óptima, es decir, aquella sobre la que los GA funcionan mejor, es aquella que tiene un alfabeto binario. La mayoría de las veces una codificación correcta es la clave de una buena resolución del problema.

El diseño de la codificación de la solución es todo un arte y no existe una única manera de codificar que trabaje bien con todos los tipos de problema. Afortunadamente, GA es robusto en el sentido de que puede devolver una solución óptima o cercana al óptimo independiente de la codificación usada. Sin embargo, en la práctica, se tiene límite en el uso de los recursos (poder de procesamiento, memoria, tiempo disponible) para resolver el problema. Aunque los GA utilizando diferentes codificaciones sean capaces de acercarse a la solución óptima, algunas son mas eficientes que otras.

En GA hay dos fuentes principales de la ineficiencia:

- Si el espacio de soluciones es altamente restrictivo, una gran porción del mismo representan soluciones ilegales. GA se pasaría demasiado tiempo evaluando cromosomas ilegales que no contribuirán al resultado final. Además los cromosomas ilegales también reducen la efectividad de la población. Como resultado, una población muy grande requiere gran cantidad de memoria.
- Soluciones que involucren una gran cantidad de esquemas son interrumpidas fácilmente por los operadores genéticos clásicos. Una vez corrompidas, pueden ser recuperados solamente por mutaciones al azar que son procesos largos y poco prometedores o por recombinaciones de los sub-esquemas.

El comportamiento de los GA según la codificación depende mucho del problema y actualmente no existe un método que guíe sobre que codificación es la mejor.

A.2.3. Función de Aptitud

Una función de aptitud usualmente está compuesta por la función de costo y por una función de castigo. La función de costo devuelve que tan buena es la solución en el dominio del problema y la función de castigo devuelve si una solución es factible y si no es factible que tanto está alejada de una solución factible. Ambas funciones juntas, usualmente combinadas en forma lineal proveen la manera de clasificar las diferentes soluciones.

La función de costo generalmente es la función objetivo del problema. La función de castigo puede no ser utilizada si los operadores genéticos no producen soluciones inválidas. De todos modos el uso de la función de castigo implica tener conocimiento del problema específico.

Una fuerte función de castigo permite rápidamente descartar los cromosomas ilegales, pero a su vez estos cromosomas pueden contener buenos esquemas, lo que puede hacer perder a la población esos esquemas al descartar los cromosomas inválidos. Sin embargo, con un castigo menor la presión por encontrar soluciones válidas es menor y la búsqueda se hace ineficiente. De modo que una buena función de castigo debe encontrar el balance entre conservar la información y la presión por la factibilidad.

A.2.4. Inicialización

Generalmente la población de cromosomas de la generación inicial es generada en forma aleatoria, pero sin embargo, también se puede utilizar un método que construya los cromosomas utilizando alguna estrategia golosa (greedy). En la inicialización, luego de generar la población, se evalúa cada uno de los cromosomas según su aptitud.

A.2.5. Reproducción

La operación de reproducción selecciona los individuos de una población a los que luego se les aplica las operaciones genéticas de manera de obtener los descendientes.

En cada iteración de GA, la población existente puede ser reemplazada por una nueva generación o algunos de los individuos pueden ser mantenidos mientras que los otros son reemplazados por nuevos. La primera forma de reproducción se llama Generational Reproduction y la otra forma se llama Steady State Reproduction.

A.2.6. Selección

La selección devuelve padres seleccionados probabilísticamente. Aunque el proceso de selección es estocástico no implica que GA utilice una dirección de búsqueda. La chance de que un cromosoma sea elegido como padre está solo determinado por su función de aptitud. Existen varias formas de realizar la selección:

- **Proporcional a la Aptitud**

  El método original para la selección de los padres es el llamado “rueda-ruleta”, utilizado originalmente por Holland que esta basado en el valor de aptitud

de los individuos. En este sentido cada individuo tiene la chance de ser elegido proporcionalmente a su aptitud. El efecto de esto depende fuertemente del rango de los valores de la aptitud en la población. Por ejemplo, si el rango de valores de los cromosomas de la población está entre 5 y 10, el cromosoma de mayor valor de aptitud tiene el doble de posibilidades de ser elegido que el que tenga el menor valor de aptitud, sin embargo si los valores están entre 1005 y 1010, todos los individuos tiene prácticamente la misma posibilidad de ser elegidos. La manera de solucionar este problema es escalar la aptitud antes de la selección [Gol89].

- **Ranking**

  La probabilidad de selección de los cromosomas está basada en el orden dentro de la población dado por su valor de aptitud [Bak85]. La ventaja de este método es que usa la información relativa de la aptitud, ignorando los valores absolutos.

- **Torneo**

  En la la selección por torneo dos individuos son seleccionados de forma aleatoria y uno de ellos (el "ganador") es seleccionado como padre. La selección del "ganador" se hace en forma aleatoria, según la probabilidad de elegir el de mejor aptitud.

A.2.7. Cruzamiento

El cruzamiento consiste en el intercambio de segmentos de los cromosomas entre si. El cruzamiento es el principal operador genético, hasta el punto que un algoritmo sin cruzamiento no es un GA. Para aplicarlo se escogen aleatoriamente dos individuos de la población y se cruzan sus respectivos materiales genéticos creando dos nuevos cromosomas descendientes de los anteriores.

Existen varios tipos de cruzamiento, figura A.2:

- Cruzamiento de un punto: Se genera un número aleatorio menor o igual que el largo del cromosoma. Luego se intercambian los bits después de la posición entre los dos padres.
- Cruzamiento de dos puntos: Es similar al anterior exceptuando que se deben seleccionar dos posiciones y solamente los bits entre las dos posiciones son intercambiados.
- Cruzamiento de n puntos: Es similar a los anteriores, excepto que se debe seleccionar n posiciones y solo los bits entre las posiciones impares y pares son intercambiados.

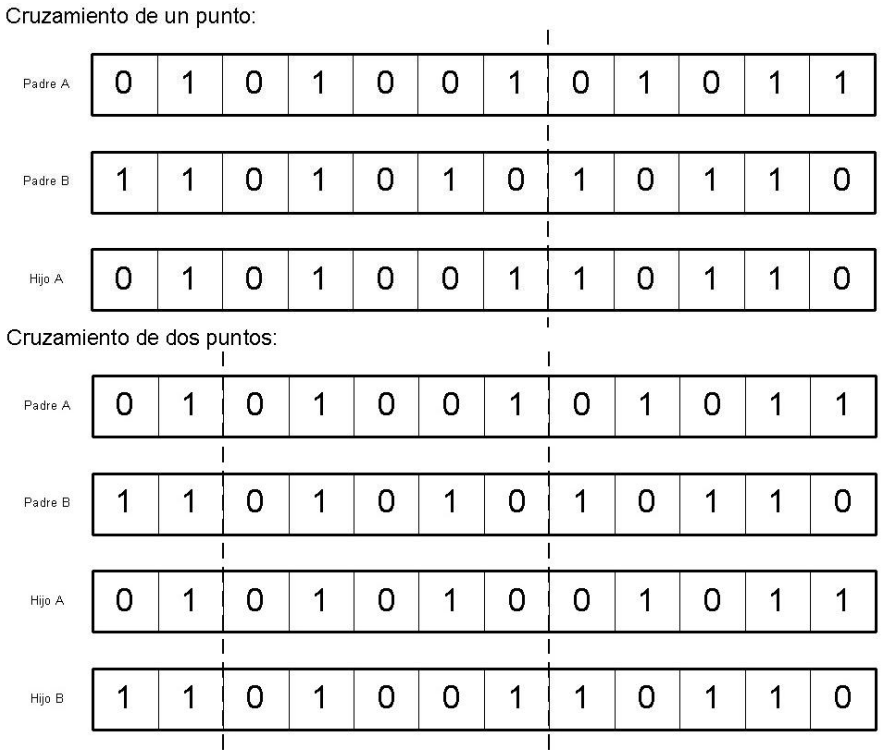


Figura A.2: Cruzamiento de Cromosomas.

- Cruzamiento Uniforme: Cada gen del primer padre tiene 0.5 de probabilidad de ser intercambiado con el correspondiente gen del segundo padre.

A.2.8. Mutación

La mutación tiene el efecto de asegurar que todos los posibles cromosomas sean alcanzables. Por ejemplo, si la primera posición de todos los cromosomas de la población inicial no tiene el valor 1, solo utilizando la operación de cruzamiento es imposible generar cromosomas con valor 1 en la primera posición.

En la mayoría de los casos las mutaciones son mortales pero, en promedio, contribuyen a la diversidad genética de la especie. En los GA, tiene el mismo papel y la frecuencia de ocurrencia es muy baja.

La mutación es un mecanismo de generación de diversidad y por tanto es solución cuando un GA está estancado, pero también puede convertir la búsqueda en una búsqueda aleatoria. Siempre es mas conveniente usar otros mecanismos para generar diversidad, como aumentar el tamaño de la población inicial o garantizar aleatoridad de la población inicial.

La manera mas común de usarla es definiendo la probabilidad de que cada gen de los cromosomas de la población sea modificado.

A.3. Búsqueda Tabú

A.3.1. Introducción General

La Búsqueda Tabú (Tabu Search, TS) es una técnica de búsqueda local diseñada para resolver problemas de optimización. Los orígenes de TS puede situarse en diversos trabajos publicados a finales de los 70' [Glo77]. Oficialmente, el nombre y la metodología fueron introducidos posteriormente por Fred Glover en 1989 y 1990 [Glo89], [Glo90] y específicamente aplicado al QAP por Skorin-Kapov [SK90] y Taillard [Tai90]. Numerosas aplicaciones han aparecido en la literatura, así como artículos y libros para difundir el conocimiento teórico del procedimiento [GL97].

TS es un procedimiento heurístico utilizado para resolver problemas de optimización de gran escala, el cual está diseñado para guiar a otros procedimientos de búsqueda local (procesos componentes) para escapar de la optimalidad local y poder explorar mejor y mas extensamente el espacio de soluciones. Tiene como antecedentes a varios métodos diseñados para cruzar fronteras de factibilidad o de optimalidad local; sistemáticamente impone y relaja restricciones para permitir la exploración de regiones de otra manera prohibidas. TS utiliza la estrategia de evitar que la búsqueda quede atrapada en un óptimo local que no sea global, para eso utiliza el concepto de memoria tratando de extraer información de lo ya sucedido y actuar en consecuencia.

A.3.2. Descripción del método

La técnica TS está basada en la noción de vecindad, dado un problema de optimización P, sea S el espacio de soluciones factibles del problema P y la función de costo f definida en S. Una función $\mathcal{N}$, que depende de la estructura del problema, que asigna a cada solución factible $s \in S$ sus vecinos $\mathcal{N}(s) \subseteq S$ y cada solución $s' \in \mathcal{N}(s)$ es llamada vecina de s.

TS comienza con una solución inicial s_i, que puede ser obtenida con otra técnica o generada en forma aleatoria, a la cual se le calcula el costo $f(s_i)$ y se explora un subconjunto V de los vecinos $\mathcal{N}(s_i)$, de estas se elige la mejor, independientemente si ésta mejora o no el valor de la función objetivo en s_i. La característica principal es precisamente la construcción de una lista tabú de movimientos, aquellos movimientos que no son permitidos en la presente búsqueda. La razón de la lista tabú es la de excluir la posibilidad de regresar a puntos visitados en alguna búsqueda anterior. Pero un movimiento permanece en la lista por un cierto número de búsquedas, de forma que se tiene una lista cíclica en donde la nueva solución es agregada y se elimina la mas vieja.

La lista tabú tiene la meta de prevenir ciclos e inducir la exploración de nuevas

regiones. Pero también una buena trayectoria de búsqueda resulta al revisitar una solución encontrada anteriormente. Ahora bien, los movimientos tabú no son inviolables para toda circunstancia. Cuando un movimiento tabú proporciona una solución mejor que cualquier otra previamente encontrada, su clasificación como tabú puede eliminarse. La condición que permite dicha eliminación se llama Criterio de Aspiración. Sea $s' \in \mathcal{N}(s)$ una solución alcanzada a través del movimiento m desde s, si $f(s') \leq A(f(s))$, s' puede ser aceptada aunque m esté en la lista tabú.

Es así como las restricciones tabú y el criterio de aspiración juegan un papel complementario en la restricción y guía del proceso de búsqueda. Las restricciones tabú permiten que un movimiento sea admisible si no está clasificado como tabú, mientras que si el criterio de aspiración se satisface, permite que un movimiento sea admisible aunque este clasificado como tabú. En forma simple TS descubre dos de sus elementos claves: la de restringir la búsqueda mediante la clasificación de ciertos movimientos como prohibidos (tabú) y el de liberar la búsqueda mediante una función de memoria de término corto que proporciona una estrategia de olvido.

La estructura genérica del algoritmo TS se muestra en el algoritmo 21.

Algorithm 21 *Metaheurística TS*

1: S_{actual} = InicializarSolución()
2: $L = \emptyset$ {Lista Tabú}
3: $S_{mejor\_global} = S_{actual}$
4: **while** criterio terminación no satisfecho **do**
5: V = GenerarVecinos($\mathcal{N}(S_{actual}), L$) $\{S \in \mathcal{N}(S_{actual}) \wedge (S \notin L \vee CriterioAspiracion)\}$
6: S_{actual}= Mejor(V)
7: $L = L \bigcup S_{actual}$
8: $S_{mejor\_global} = Mejor(S_{actual}, S_{mejor\_global})$
9: **end while**

El procedimiento *InicializarSolución* genera la solución inicial, que generalmente es una solución generada en forma aleatoria o mediante un método constructivo, por ejemplo puede ser un procedimiento goloso (greedy). También se puede utilizar como solución inicial el resultado de la aplicación de otro método conocido, o el uso de una solución conocida. *GenerarVecinos* genera el conjunto de soluciones vecinas que no están en la lista tabú o si estando en la lista tabú cumplen el criterio de aspiración.

Los parámetros que permiten controlar el algoritmo son:

- La cardinalidad del conjunto de vecinos V, y
- el largo de la lista tabú.

A.4. Optimización por Colonia de Hormigas

A.4.1. Colonia de Hormigas

Las hormigas son insectos sociales que viven en colonias y debido a su interacción colaborativa son capaces de realizar tareas complejas que no pueden realizar en forma individual. Un muy interesante aspecto que poseen varias especies de hormigas es la habilidad de encontrar el camino más corto entre el hormiguero y las fuentes de alimentos.

Mientras caminan entre el hormiguero y la fuente de alimentos, esas especies de hormigas depositan una sustancia química llamada feromona. Si no existe rastro de feromona las hormigas se mueven esencialmente en forma aleatoria, pero en presencia del mismo las hormigas tienen tendencia de seguir los rastros ricos en feromona. En efecto, experimentos realizados por biólogos [PDG87], [GAD89] demostraron que las hormigas probabilísticamente prefieren caminos que estén marcados con una gran concentración de feromona. En la práctica, la elección entre diferentes caminos se da en la intersección de los mismos, entonces las hormigas eligen el camino a seguir por una decisión probabilística basado en la cantidad de feromona depositada por las anteriores hormigas. De modo que la probabilidad de que una hormiga siga un camino es proporcional al número de hormigas que previamente han pasado por ese camino. También son capaces de adaptarse a los cambios en el ambiente, por ejemplo, encontrando un nuevo camino debido a la interrupción por algún obstáculo.

Inicialmente no existe rastro de feromona que guíe a las hormigas hacia la fuente de alimentos, por lo cual abandonan el hormiguero en cualquier dirección y lentamente van cubriendo la totalidad del espacio en donde se mueven, figura A.3. Cuando las primeras hormigas encuentran la fuente de alimentos comienzan a depositar cantidades de feromona y tratan de regresar al hormiguero. Como en principio no existe rastro de feromona, estas hormigas intentan regresar moviéndose esencialmente en forma aleatoria. Eventualmente alguna hormiga logra encontrar el hormiguero y completa un rastro de feromona desde el hormiguero a la fuente, figura A.4. Rápidamente las hormigas convergen a un solo camino, el camino mas corto entre el hormiguero y la fuente de alimentos, figura A.5. Si un obstáculo se interpone cortando el camino de las hormigas, eventualmente descubrirán el nuevo camino más corto esquivando el obstáculo, figura A.6.

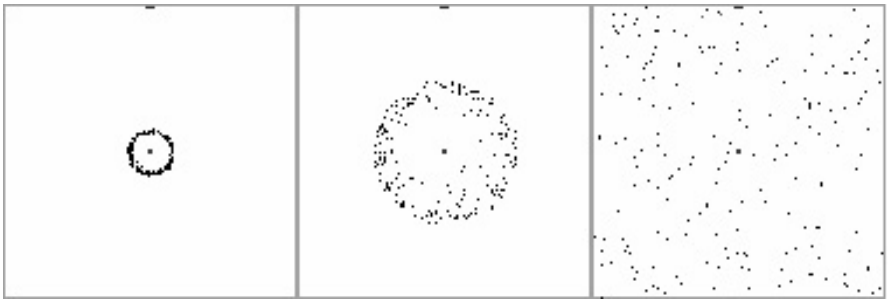


Figura A.3: Las hormigas abandonan el hormiguero.

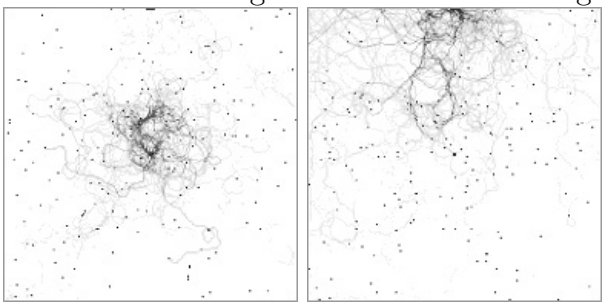


Figura A.4: Rastros de feromona de la fuente y del hormiguero.

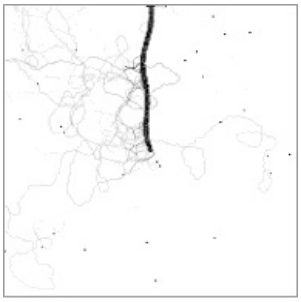


Figura A.5: Ruta más corta.

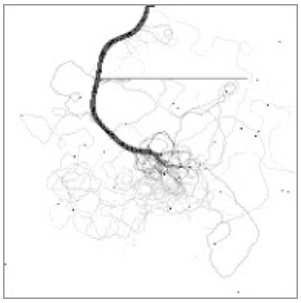


Figura A.6: Ruta alrededor del obstáculo.

Para entender el mecanismo, se define un caso mas restringido en donde existen dos posibles rutas alrededor del obstáculo: Hormiguero-ABD-Fuente y Hormiguero-ACD-Fuente, que tienen largo 4 y 6 respectivamente, figura A.7. Cada hormiga se mueve una unidad de distancia por unidad de tiempo. Inicialmente no existen rastros de feromona. En:

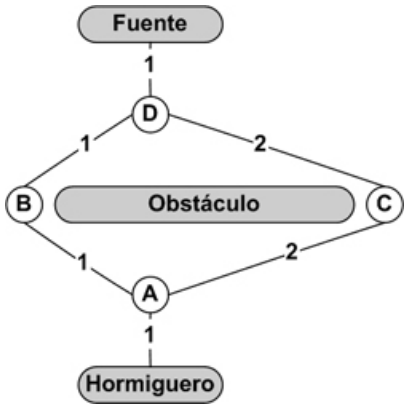


Figura A.7: Movimientos de las hormigas.

- t = 0, aún no existe rastro de feromona y las hormigas todavía están en el hormiguero.
- t = 1, 20 hormigas dejaron el hormiguero y llegan a A. Eligen con igual probabilidad tomar hacia la derecha o izquierda, entonces en promedio 10 van hacia B y las otras 10 van a C.
- t = 4, las primeras hormigas llegan a la fuente y comienzan a volver.
- t = 5, los dos grupos de 10 hormigas se encuentran en D, solo que una van y otras vienen desde la fuente. En este momento el rastro de feromona en BD es el mismo que en CD, debido a que 10 hormigas pasaron por ambos. Entonces de las 10 que vuelven al hormiguero 5 eligen DB y las otras 5 eligen DC.
- t = 8, las primeras 5 hormigas llegan al hormiguero.
- t = 9, las 5 hormigas que llegaron al hormiguero vuelven a A. El rastro en AB es de 20 y el AC es de 15. De modo que la mayoría de las hormigas elegirán el camino AB y aumentarán aún más el rastro para esa alternativa.

Al continuar este proceso, la diferencia en la cantidad de feromona entre las dos posibles rutas seguirá en aumento y la mayoría de las hormigas tomarán el camino más corto.

A.4.2. Metaheurísticas ACO

Los algoritmos de Optimización por Colonia de Hormigas (Ant Colony Optimization, ACO) utilizan una analogía de las colonias de hormigas reales para resolver los problemas de optimización combinatoria. Los algoritmos ACO están basados en una colonia de hormigas artificiales, agentes simples que trabajan en forma cooperativa y se comunican a través de los rastros artificiales de feromona. Básicamente, en cada iteración del algoritmo, cada hormiga construye una solución al problema viajando a través del grafo definido para el problema $G = (N, A)$. Cada arista del grafo $a_{rs} \in A$, representando los posible pasos que la hormiga puede tomar, tiene asociada dos tipos de información que guían los movimientos de las hormigas [CHS02]:

- Información Heurística, que mide la preferencia de moverse del nodo r al nodo s, a través de la arista a_{rs}. Es denotado por η_{rs} y se le llama visibilidad. Esta información no es modificada por las hormigas durante la ejecución del algoritmo y depende del problema en sí.
- Información del Rastro (artificial) de Feromona, que mide que tan bueno es elegir el movimiento. Esta información es modificada durante la ejecución del algoritmo dependiendo de la solución encontrada por las hormigas. Es denotada por τ_{rs} y se le llama intensidad del rastro.

Por un lado los algoritmos utilizan una abstracción de la realidad de las hormigas naturales, pero por otro lado incluyen características que no se encuentran en las mismas o bien adaptan las ya existentes en la naturaleza (uso de la información heurística, memoria, la evaporación del rastro de feromona y también la posibilidad de realizar acciones globales de forma opcional). La evaporación de los rastros de feromona es el proceso que hace que disminuya la intensidad del mismo a través del tiempo. Desde un punto de vista práctico, la evaporación de los rastros de feromona es necesaria para evitar una rápida convergencia del algoritmo a una región óptima local. Esto implementa una manera de brindarle cierta característica de "olvido" al algoritmo que le permita explorar nuevas áreas en el espacio de búsqueda. Las acciones globales pueden ser usadas para implementar acciones centralizadas que no pueden ser realizadas por una simple hormiga. Por ejemplo, la activación de un proceso de búsqueda local o el uso de información general que pueda ser usada en decidir si es necesario o no depositar feromona adicional de manera de ayudar al algoritmo a tener una perspectiva mas general y no solo local.

Tipos de Problemas para ACO

Los tipos de problemas que pueden ser resueltos por las hormigas artificiales pertenecen al grupo de los problemas de trayectoria mas corta. Consideremos el problema

de minimizar (Γ, C, Ω), donde Γ es el conjunto de soluciones factibles, C es la función objetivo que asigna a cada solución $S \in \Gamma$ un valor (costo) $C(S)$ y Ω es el conjunto de restricciones. El objetivo es encontrar una solución óptima global $S_{opt} \in \Gamma$ de costo mínimo que satisfaga las restricciones Ω. La representación del problema de optimización combinatoria (Γ, C, Ω) que puede ser resuelto por ACO puede tener las siguientes características [DC99], [DS02], [CHS02]:

- Un conjunto finito de componentes $N = \{n_1, n_2, ..., n_l\}$
- Un conjunto finito A de conexiones/transiciones entre los elementos de N, $A = \{a_{rs}/n_r \in N \wedge n_s \in N\}$.
- Un conjunto de restricciones Ω.
- Los estados del problema son definidos como secuencias $\delta =< n_r, n_s, ..., n_u, ... >$ ($< r, s, ..., u, ... >$ para simplificar) de los elementos de N. Si Δ es el conjunto de todos las posibles secuencias δ, se denota como $\tilde{\Delta}$ el conjunto de (sub)secuencias factibles con respecto a las restricciones Ω. Los elementos en $\tilde{\Delta}$ definen los estados factibles del problema. $|\Delta|$ es el largo de la secuencia Δ, el número de componentes de la secuencia.
- Una estructura de vecindad definida como sigue: δ_2 es vecino de δ_1 si (i) δ_1 y δ_2 pertenecen a Δ, (ii) el estado δ_2 puede ser alcanzado por δ_1 en un movimiento lógico. Si n_r es el último componente de la secuencia δ_1, debe existir un componente $n_s \in N$ de modo que $\delta_2 =< \delta_1, n_s >$ y existe una transición valida entre n_r y n_s. El vecindario factible de δ_1 es el conjunto que contiene todas las secuencias de $\delta_2 \in \tilde{\Delta}$; si $\delta_2 \notin \tilde{\Delta}$, se dice que δ_2 está en el vecindario no factible de δ_1.
- Una solución S es un elemento de $\tilde{\Delta}$ verificando todas las restricciones Ω del problema.
- Un costo $C(S)$ asociado a cada solución $S \in \tilde{\Delta}$.
- En algunos casos, el costo o una estimación del costo puede ser asociado a los estados.

Según lo dicho, todas las características anteriores de los problemas de optimización combinatoria pueden ser representados de la forma de un grafo con pesos $G = (N, A)$, donde A es el conjunto de aristas que conectan el conjunto de nodos N. El grafo G es también llamado grafo de construcción G. Las soluciones del problema de optimización pueden ser expresadas en términos de caminos factibles en el grafo G. Los algoritmos ACO pueden ser usados para encontrar el camino (secuencia) de costo mínimo respecto a las restricciones Ω para el grafo G.

De modo que se tiene que:

- n_r son los nodos del grafo,
- los estados δ (y la solución S) corresponden con el camino en el grafo, secuencias de nodos o aristas,
- las aristas del grafo a_{rs} definen la estructura del vecindario. $\delta_2 =< \delta_1, s >$ es un vecino de δ_1 si el nodo r es el último componente de δ_1 y la arista a_{rs} existe en el grafo,
- puede haber un costo de transición c_{rs} asociado a cada arista, y
- los nodos y/o aristas tienen asociado un rastro de feromona τ y una visibilidad η.

La Hormiga Artificial

Las hormigas artificiales son simples agentes que tratan de construir soluciones factibles del problema utilizando la información heurística y los rastros de feromona. Sin embargo, si es necesario, pueden construir soluciones no factibles que son penalizadas dependiendo de la cantidad de infactibilidad.

Las hormigas artificiales tienen las siguientes características generales [DC99], [DS02], [CHS02]:

- Aunque cada hormiga es capaz de encontrar una solución (probablemente una mala solución), soluciones de buena calidad pueden solo surgir como resultado de la interacción colectiva de varias de ellas.
- Cada hormiga solamente utiliza su información y la información local del nodo que está visitando.
- Se comunican con las otras hormigas solamente en forma indirecta, mediante la información del rastro de feromona depositado por cada una de ellas.

También se tiene para cada hormiga k las siguientes propiedades [DC99], [DS02], [CHS02]:

- Busca la solución factible S^k de mínimo costo para el problema que está resolviendo.

- Tiene una memoria L^k que se usa para guardar información sobre el camino seguido hasta el momento y L^k almacena la secuencia generada. Esa memoria puede ser usada para: (i) construir soluciones factibles, (ii) evaluar la solución generada, y (iii) reconstruir el camino seguido por la hormiga, que generalmente se usa para actualizar los rastro de feromona.

- Tiene un estado inicial $\delta^k_{inicial}$, que usualmente corresponde a la secuencia unitaria y una o más condiciones de terminación t asociada.

- Comienza en el estado inicial $\delta^k_{inicial}$ y se mueve a través de los estados vecinos factibles, construyendo incrementalmente su solución asociada. El procedimiento de construcción se detiene cuando por lo menos alguna de las condiciones de terminación t es cumplida.

- Cuando está en el estado $\delta_r =< \delta_{r-1}, r >$ (se encuentra en el nodo r y previamente siguió la secuencia δ_{r-1}), se puede mover a cualquier nodo s de su vecindario factible $\mathcal{N}^k_r$, definido como $\mathcal{N}^k_r = \{s/a_{rs} \in A \wedge < \delta_r, s > \in \tilde{\Delta}\}$.

- Cuando se encuentra en el nodo n_r puede moverse al nodo n_s si $s \in \mathcal{N}^k_r$ aplicando una regla de transición probabilística.

- La regla de transición es definida en (i) función del rastro de feromona y de los valores heurísticos, (ii) la memoria L^k almacenando la historia pasada, y (iii) las restricciones del problema Ω.

- Cuando, durante el procedimiento de construcción, se mueve del nodo r al s, ella puede actualizar el rastro de feromona τ_{rs} asociado a la arista a_{rs}. Este proceso es llamado "online step-by-step pheromone trail update".

- Una vez que la solución es construida, puede rehacer el camino y actualizar el rastro de feromona de las aristas visitadas. Este proceso es llamado "online delayed pheromone trail update".

Similitudes y diferencias entre hormigas naturales y artificiales

Las hormigas reales y artificiales comparten un número de características. Las mas importantes son las siguientes [DCG99], [CHS02]:

- Como las hormigas reales, las hormigas artificiales forman una colonia de individuos que interactúan y colaboran para encontrar "buenas" soluciones al problema.

- Modifican su "entorno" a través de una comunicación basada en la feromona. En el caso de las hormigas artificiales, el rastro artificial de feromona es información numérica disponible localmente y es el canal de comunicación que utilizan.

- Comparten tareas en común: la búsqueda del camino mas corto (la construcción iterativa de la solución de costo mínimo) de un origen, el hormiguero, a un destino final, la fuente de alimentos.
- Las hormigas artificiales construyen la solución iterativamente aplicando políticas de transición estocásticas locales, de la misma manera que las naturales.

Sin embargo, estas características, por si sola, no garantizan el desarrollo de algoritmos eficientes para los problemas de optimización combinatoria. Por lo tanto, las hormigas artificiales tienen capacidades adicionales:

- Viven en un mundo discreto y sus movimientos consisten en transiciones desde estados discretos a estados discretos.
- Las hormigas artificiales pueden hacer uso de la información heurística (y no solo la información del rastro de feromona) en la política de transición estocástica que ellas apliquen.
- Tienen memoria para almacenar el camino seguido.
- La cantidad de feromona depositadas por la hormigas artificiales es en función de la calidad de las soluciones encontradas por cada una de ellas. La mayor diferencia consiste en el momento en que se deposita la feromona. Usualmente las hormigas artificiales depositan feromona luego de completada la construcción de la solución.
- La evaporación de feromona en los algoritmos ACO difiere con el de la naturaleza, desde que la inclusión de un mecanismo de evaporación es la clave para que el algoritmo escape de óptimos locales, en el momento e intensidad con que se realiza. La evaporación de la feromona permite a la colonia de hormigas artificiales olvidarse "suavemente" de la historia pasada y directamente dirigir la búsqueda hacia nuevas regiones de búsqueda y así evitar la prematura convergencia del algoritmo a óptimos locales. Es interesante hacer notar, que aunque los rastros reales de feromona se evaporen, esa evaporación no juega un papel importante en encontrar el camino mas cortos con las hormigas reales [DS01].
- De manera de mejorar la eficiencia y eficacia de los algoritmos ACO, los mismos son extendidos con capacidades adicionales, por ejemplo, optimización local [DG97b], [SH97], de manera de aumentar la eficiencia de los algoritmos.

Modo de Operación y Estructura Genérica

El modo de operación básico de los algoritmos ACO es el siguiente [DC99], [DCG99], [CHS02]: las m hormigas (artificiales) de la colonia se mueven concurrentemente y

asincrónicamente a través de los estados adyacentes del problema. Estos movimientos son hechos de acuerdo a la regla de transición que esta basada en la información local disponible de los componentes (nodos). Esa información local comprende información heurística y memoria (rastro de feromona) que guía la búsqueda. Durante los movimientos en el grafo las hormigas incrementalmente construyen las soluciones. Opcionalmente, las hormigas pueden liberar feromona cada vez que cruzan una arista mientras construyen la solución ("online step-by-step pheromone trail update"). Una vez que cada hormiga genera una solución, es evaluada y puede depositarse una cierta cantidad de feromona que es en función de la calidad de la solución encontrada ("online delayed pheromone trail update"), esta información guía la búsqueda para las otras hormigas de la colonia en el futuro.

Las acciones globales son opcionales y permiten realizar tareas desde una perspectiva global que no tienen las hormigas en su perspectiva local. Tanto para la liberación de feromona para las aristas visitadas por la hormiga que brinda la mejor solución o como realizar una búsqueda local a la solución generada antes de actualizar los rastros de feromona, en ambos casos el proceso llamado "online delayed pheromone trail update" es sustituido por un procedimiento llamado "offline pheromone trail update". La evaporación del rastro, desde un punto de vista práctico, es necesario para evitar una rápida convergencia del algoritmo en una región óptima local, implementando la posibilidad de permitirle al algoritmo cierto "olvido" favoreciendo la exploración de nuevas regiones en el espacio de búsqueda.

La estructura genérica del algoritmo ACO se muestra en los algoritmos 22, 23 y 24 [DC99], [DCG99], [CHS02].

Algorithm 22 *Metaheurística ACO*

```
1: InicializarParámetros()
2: while criterio terminación no satisfecho do
3:    schedule activities
4:    GeneraciónHormigasActividad()
5:    EvaporaciónFeromona()
6:    AccionesGlobales() {opcional}
7:    end schedule activities
8: end while
```

Algorithm 23 *GeneraciónHormigasActividad()*

```
1: for k = 1 to m {número de hormigas} do
2:    NuevaHormiga(k)
3: end for
```

Algorithm 24 *NuevaHormiga(k)*

```
1: InicializarHormiga(k)
2: L^k = ActualizarMemoriaHormiga()
3: while estado_actual ≠ estado_destino do
4:    P^k = ComputarProbabilidadTransición(A, L^k, Ω)
5:    estado_siguiente = AplicarDecisiónHormiga(P^k, Ω)
6:    MoverSiguienteEstado(estado_siguiente)
7:    if online step by step pheromone update then
8:       DepositarFeromonaAristaVisitada()
9:    end if
10:   L^k = ActualizarEstadoInterno()
11: end while
12: if online delayed pheromone update then
13:   for all arista_visitada do
14:      DepositarFeromonaAristaVisitada()
15:   end for
16: end if
17: LiberarRecursosHormiga(k)
```

El primer paso del algoritmo es la inicialización de los parámetros, entre otros, el valor del rastro inicial de feromona asociado a cada transición, τ_0, el cual es un valor positivo pequeño que típicamente es el mismo para todos las conexiones/componentes, el número de hormigas en la colonia m y el peso definido entre la información heurística y el rastro de feromona en la regla de probabilidad de transición. Los tres componentes básicos son (i) la generación y operación de las hormigas artificiales, (ii) la evaporación de feromona y (iii) las acciones globales que dependen del algoritmo ACO específico, por ejemplo, cuando y donde la feromona es depositada y realizar una búsqueda local. La **schedule activities** no especifica como las tres actividades son programadas y sincronizadas. En otras palabras, no especifica la manera que deben ejecutarse en un ambiente paralelo o si algún tipo de sincronización es necesaria. El diseñador es libre de especificar la manera que estos procedimientos deben interactuar.

El procedimiento *ActualizarMemoriaHormiga* especifica el estado inicial de donde cada hormiga comienza su recorrido y lo almacena en la memoria L^k. Finalmente, el procedimiento *ComputarProbabilidadTransición* y el procedimiento *AplicarDecisiónHormiga* consideran el estado actual de la hormiga, el valor de la feromona visible en cada nodo y las restricciones del problema Ω que establece el proceso de transición probabilístico a otros estados factibles.

A.4.3. Ant System

Ant System (AS) [DMC96], desarrollado por Dorigo, Maniezzo y Colorni en 1991 [DMC91], [DMC92], fue el primer algoritmo ACO y fue aplicado inicialmente al Pro-

blema del Viajante de Comercio (Traveling Salesman Problem). Inicialmente, tres diferentes versiones de AS fueron propuestas: ant-density, ant-quantity y ant-cycle. En ant-density y ant-quantity las hormigas modifican el rastro de feromona mientras construyen la solución, aplicando un procedimiento "online step-by-step pheromone update", con la diferencia que en ant-density la cantidad de feromona depositada es constante y en ant-quantity la cantidad de feromona depositada depende directamente de la visibilidad η_{rs}. Finalmente, en ant-cycle, el rastro de feromona es actualizado una vez que todas las hormigas hayan construido su solución y la cantidad depositada por cada hormiga depende de la solución encontrada, aplicando un procedimiento "online delayed pheromone trail update".

Experimentos preliminares para un conjunto de problemas [DMC92], [Dor92], [DMC96] demostraron que la performance del algoritmo ant-cycle es mucho mejor que los otros dos algoritmos. Como consecuencia, las investigaciones en AS se concentraron en entender mejor las características de ant-cycle, el cual es ahora conocido simplemente como Ant System, mientras que los otros dos algoritmos fueron abandonados.

En AS, en cada paso de la construcción de la solución, una hormiga k elige moverse al siguiente nodo con una probabilidad dada por:

$$p_{rs}^k = \begin{cases} \frac{[\tau_{rs}]^\alpha . [\eta_{rs}]^\beta}{\sum_{u \in \mathcal{N}_r^k} [\tau_{ru}]^\alpha . [\eta_{ru}]^\beta} & \text{si } s \in \mathcal{N}_r^k \\ 0 & \text{sino.} \end{cases}$$

donde $\mathcal{N}_r^k$ es el conjunto de nodos vecinos factibles al nodo r para la hormiga k y $\alpha, \beta \in R$ son dos parámetros que determinan la importancia relativa del rastro de feromona y la información heurística. Cada hormiga k almacena la secuencia seguida en su memoria L^k que es utilizada para determinar el vecindario $\mathcal{N}_r^k$ en cada etapa de la construcción, de modo que $\mathcal{N}_r^k = \{s / a_{rs} \in A \wedge s \notin L^k \wedge < S^k, s > \text{cumple restricciones } \Omega\}$.

El rol de cada parámetro α y β es el siguiente: si $\alpha = 0$, aquellos nodos con mejor preferencia heurística tienen mayor probabilidad de ser elegidos, haciendo que el algoritmo se comporte como un clásico algoritmo probabilístico goloso (greedy) (con multiples puntos de inicio en caso de que las hormigas sean colocadas en diferentes nodos en el comienzo de cada iteración). Sin embargo, si $\beta = 0$ solamente el rastro de feromona es considerado para guiar el proceso constructivo, el cual causa una rápido estancamiento del algoritmo. Esto hace que las hormigas construyan la misma solución, usualmente una solución óptima local. De manera que es necesario establecer un balance apropiado entre la importancia de la información heurística y la información del rastro de feromona.

La construcción de cada solución S^k termina luego de que cada hormiga k ha completado su recorrido.

El rastro de feromona es actualizado una vez que todas las hormigas han terminado

de construir su solución. Primero, el rastro de feromona asociado a cada arista es evaporado reduciéndolos a todos por un factor constante:

$$\tau_{rs} = (1 - \rho).\tau_{rs} \quad \forall a_{rs}$$

donde $0 < \rho \leq 1$ es el coeficiente de evaporación del rastro, $(1 - \rho)$ es el coeficiente de persistencia del rastro. Y la cantidad de feromona liberada es $\Delta\tau_{rs}^k = f(C(S^k))$ que depende de la calidad de la solución S^k obtenida por la hormiga k.

$$\tau_{rs} = \tau_{rs} + \Delta\tau_{rs}^k \quad \forall a_{rs} \in S^k, \forall k$$

El parámetro ρ es usado para limitar la acumulación de feromona y posibilitar al algoritmo "olvidar" las malas decisiones previas. En una arista que no es visitada por las hormigas la intensidad de su rastro de feromona decrece en forma exponencial con el número de iteraciones.

Generalmente la cantidad de feromona depositada por cada hormiga es dada por:

$$\Delta\tau_{rs}^k = \begin{cases} 1/C(S^k) & \text{si } a_{rs} \text{ es usada por la hormiga } k \\ 0 & \text{sino.} \end{cases}$$

Los creadores de AS también propusieron una mejora a este algoritmo llamada estrategia elitista [DMC96]. En $\text{AS}_{elitista}$ solo la hormiga que obtenga la mejor solución actualiza los rastros de feromona. La cantidad de feromona depositada depende de la calidad de esa mejor solución encontrada.

$$\tau_{rs} = \tau_{rs} + e.f(C(S_{mejor\_global})) \quad \forall a_{rs} \in S_{mejor\_global}$$

donde S_{mejor_global} es la mejor solución obtenida por las hormigas y e es el número de hormigas elitistas.

Para terminar la descripción de AS, la composición del procedimiento *NuevaHormiga* se muestra en el algoritmo 25 y el procedimiento *EvaporaciónFeromona* en el algoritmo 26. Los parámetros que controlan el algoritmo son:

- la cantidad de hormigas m utilizadas en cada iteración,
- el rastro inicial de cada arista τ_0,
- el coeficiente de evaporación de los rastros ρ,
- la ponderación del rastro α, y
- la ponderación de la visibilidad β.

Algorithm 25 *NuevaHormiga(k)*

1: r = GenerarEstadoInicial()
2: $S^k = r$
3: $L^k = r$
4: **while** estado_actual $\neq$ estado_destino **do**
5: **for all** $s \in \mathcal{N}_r^k$ **do**
6: $p_{rs}^k = \frac{[\tau_{rs}]^\alpha . [\eta_{rs}]^\beta}{\sum_{u \in \mathcal{N}_r^k} [\tau_{ru}]^\alpha . [\eta_{ru}]^\beta}$
7: **end for**
8: estado_siguiente = AplicarDecisiónHormiga($P, \mathcal{N}_r^k$)
9: r = estado_siguiente
10: $S^k =< S^k, r >$
11: $L^k = L^k \bigcup r$
12: **end while**
13: {el procedimiento EvaporaciónFeromona() realiza la evaporación en cada arista $a_{rs} : \tau_{rs} = (1-\rho).\tau_{rs}$ y para cada solución actualiza el rastro $\tau_{rs} = \tau_{rs} + f(C(S^k))$}
14: LiberarRecursosHormiga(k)

Algorithm 26 *EvaporaciónFeromona()*

1: **for all** a_{rs} **do**
2: $\tau_{rs} = (1 - \rho).\tau_{rs}$
3: **end for**
4: **for** $k = 1$ **to** m {número de hormigas} **do**
5: **for all** $a_{rs} \in S^k$ **do**
6: $\tau_{rs} = \tau_{rs} + f(C(S^k))$
7: **end for**
8: **end for**

A.4.4. Ant Colony System

Ant Colony System (ACS) [DG97b] fue introducido para mejorar la performance de AS. ACS difiere en tres aspectos principales de AS: (i) la regla de transición provee un mejor balance entre la exploración de nuevas aristas y la explotación del conocimiento acumulado del problema, utilizando una regla de transición llamada "pseudo-random proportional rule", (ii) la regla de actualización global del rastro de feromona es aplicada solamente a las aristas de la mejor solución y (iii) mientras las hormigas construyen una solución una regla de actualización local de feromona "local pheromone updating rule" es aplicada.

- (i) ACS utiliza una regla de transición diferente, la cual es llamada "pseudo-random proportional rule". Dada una hormiga k ubicada en el nodo r, $q_0 \in [0,1]$ es un parámetro y q un valor aleatorio en $[0,1]$. El siguiente nodo s es elegido de acuerdo a la siguiente distribución de probabilidad:

  si $q \leq q_0$,

$$p_{rs}^k = \begin{cases} 1 & \text{si } s = arg\,\text{máx}_{u \in \mathcal{N}_r^k}\{\tau_{ru}.\eta_{ru}^\beta\} \\ 0 & \text{sino.} \end{cases}$$

  si $q > q_0$,

$$p_{rs}^k = \begin{cases} \frac{[\tau_{rs}].[\eta_{rs}]^\beta}{\sum_{u \in \mathcal{N}_r^k}[\tau_{ru}].[\eta_{ru}]^\beta} & \text{si } s \in \mathcal{N}_r^k \\ 0 & \text{sino.} \end{cases}$$

  Cuando $q \leq q_0$, se explota el conocimiento disponible, eligiendo la mejor opción con respecto a la información heurística y al rastro de feromona. Sin embargo, si $q > q_0$, se aplica una exploración controlada, como en AS. En definitiva la regla establece un intercambio entre la exploración de nuevas conexiones y la explotación de la información disponible hasta el momento. El parámetro q_0 determina la importancia entre exploración y explotación.

- (ii) En ACS solo la hormiga que construye la mejor solución global es habilitada a depositar feromona. El rastro de feromona es actualizado en todas las aristas usadas por la mejor hormiga como sigue:

$$\tau_{rs} = (1-\rho).\tau_{rs} + \rho.f(C(S_{mejor\_global})) \quad \forall a_{rs} \in S_{mejor\_global}$$

  Adicionalmente se puede aplicar un algoritmo de búsqueda local para mejorar la solución antes de actualizar los rastros de feromona.

- (iii) Adicionalmente a la actualización global de los rastros de feromona, en ACS las hormigas utilizan una regla de actualización local que se aplica inmediatamente luego de que la hormiga cruza una arista:

$$\tau_{rs} = (1-\varphi).\tau_{rs} + \varphi.\tau_0$$

  donde $\varphi \in (0,1]$ es otro parámetro de evaporación del rastro. Como se puede ver, la regla de actualización "online step-by-step trail update rule" incluye evaporación y persistencia del rastro. El efecto de la actualización local del rastro es hacer a una arista ya elegida menos atractiva para las siguientes hormigas. De este modo la exploración de aristas aún no visitadas es incrementada.

Los procedimientos *NuevaHormiga* y *AccionesGlobales* (el cual en este caso interactúa con el procedimiento *EvaporaciónFeromona*) para ACS se muestran en los algoritmos 27 y 28. Los parámetros que controlan el algoritmo son:

- la cantidad de hormigas m utilizadas en cada iteración,
- el rastro inicial de cada arista τ_0,
- el coeficiente de evaporación de los rastros ρ,
- el coeficiente de evaporación local de los rastros φ,
- la probabilidad q_0, y
- la ponderación de la visibilidad β.

Algorithm 27 *NuevaHormiga(k)*

r = GenerarEstadoInicial()
$S^k = r$
$L^k = r$
while estado_actual $\neq$ estado_destino **do**
  for all $s \in \mathcal{N}_r^k$ **do**
    $b_{rs} = \tau_{rs}.\eta_{rs}^{\beta}$
  end for
  q = GenerarValorRandómico$(0, 1)$
  if $q \leq q_0$ **then**
    estado_siguiente = $\max(b_{rs}, \mathcal{N}_r^k)$
  else
    for all $s \in \mathcal{N}_r^k$ **do**
      $p_{rs}^k = \frac{b_{rs}}{\sum_{u \in \mathcal{N}_r^k} b_{ru}}$
    end for
    estado_siguiente = AplicarDecisiónHormiga$(P, \mathcal{N}_r^k)$
  end if
  r = estado_siguiente
  $S^k =< S^k, r >$
  $\tau_{rs} = (1 - \varphi).\tau_{rs} + \varphi.\tau_0$
  $L^k = L^k \bigcup r$
end while
LiberarRecursosHormiga(k)

Algorithm 28 *AccionesGlobales()*

for all S^k **do**
  BúsquedaLocal(S^k) {opcional}
end for
S_{mejor_actual} = MejorSolución(S^k)
if Mejor($S_{mejor\_actual}, S_{mejor\_global}$) **then**
  $S_{mejor\_global} = S_{mejor\_actual}$
end if
for all $a_{rs} \in S_{mejor\_global}$ **do**
  {el procedimiento EvaporaciónFeromona() es sustituido por lo siguiente}
  $\tau_{rs} = (1 - \rho).\tau_{rs} + \rho.f(C(S_{mejor\_global}))$
end for

A.4.5. $\mathcal{MAX} - \mathcal{MIN}$ Ant System

$\mathcal{MAX} - \mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) [SH96], [SH00], desarrollado por Stützle y Hoos en 1996, es uno de las mejores extensiones de AS. La solución en $\mathcal{MM}$AS es construida exactamente de la misma manera que en AS y extiende el AS básico en los siguientes aspectos:

- (i) Se aplica una regla de actualización offline de los rastros de feromona ("offline pheromone trail update"), similar a ACS. Luego de que todas las hormigas han construido una solución, primero todos los rastros de feromona se evaporan:

  $$\tau_{rs} = (1 - \rho).\tau_{rs} \quad \forall a_{rs}$$

  y luego para la hormiga que obtiene la mejor solución se actualiza el rastro:

  $$\tau_{rs} = \tau_{rs} + \rho.f(C(S_{mejor})) \quad \forall a_{rs} \in S_{mejor}$$

  La mejor hormiga que es habilitada a actualizar los rastros de feromona puede ser la mejor de las m hormigas de cada iteración o la hormiga que ha construido la mejor solución global hasta ese momento. Resultados experimentales [Stü98] han demostrado que la mejor performance se obtiene incrementando gradualmente la frecuencia de elegir la mejor solución global para la actualización del rastro.

  En $\mathcal{MM}$AS típicamente la solución obtenida por una hormiga es mejorada utilizando optimización local antes de actualizar el rastro de feromona.

- (ii) Los posibles valores de los rastros de feromona están limitados al rango $[\tau_{min}, \tau_{max}]$, de manera de que la chance de que el algoritmo se estanque decrezca dando a cada conexión alguna, aunque muy pequeña, probabilidad de ser elegido.

En la práctica, existen heurísticas para fijar τ_{min} y τ_{max}. Primero, se puede demostrar que, debido a la evaporación, el valor máximo posible del rastro de feromona está limitado a $\tau^*_{max} = 1/\rho.C(S^*))$, donde S^* es la solución óptima. Basados en este resultado la mejor solución global puede ser usada para estimar τ_{max} reemplazando S^* por S_{mejor_global} en τ^*_{max}. Para τ_{min} es suficiente con elegir un valor constante menor que τ_{max} [SH00].

Como una manera de incrementar la exploración del espacio de soluciones, $\mathcal{MM}$AS también ocasionalmente re-inicializa los rastros de feromona.

- (iii) El rastro de feromona es inicializado con un valor máximo estimado obtenido de la ecuación τ^*_{max}, tomando S^* igual a una solución construida con una heurística golosa (greedy).

  Esta inicialización hace una diferencia sustancial del algoritmo, dado que al principio en las primeras iteraciones la diferencia relativa entre los rastros de feromona no es muy marcada, que es diferente cuando la inicialización del rastro de feromona se hace con un valor pequeño.

El procedimiento *AccionesGlobales* para $\mathcal{MM}$AS se muestra en el algoritmo 29. Los parámetros que controlan el algoritmo son:

- la cantidad de hormigas m utilizadas en cada iteración,
- el rastro mínimo de cada arista τ_{min},
- el rastro máximo de cada arista τ_{max},
- el coeficiente de evaporación de los rastros ρ,
- la ponderación del rastro α, y
- la ponderación de la visibilidad β.

Algorithm 29 *AccionesGlobales()*

for all S^k **do**
  BúsquedaLocal(S^k) {opcional}
end for
S_{mejor_actual} = MejorSolución(S^k)
if Mejor($S_{mejor\_actual}, S_{mejor\_global}$) **then**
  $S_{mejor\_global} = S_{mejor\_actual}$
end if
S_{mejor} = Decisión($S_{mejor\_actual}, S_{mejor\_global}$)
{el procedimiento EvaporaciónFeromona() es sustituido por lo siguiente}
for all a_{rs} **do**
  $\tau_{rs} = (1 - \rho).\tau_{rs}$
end for
for all $a_{rs} \in S_{mejor}$ **do**
  $\tau_{rs} = \tau_{rs} + f(C(S_{mejor}))$
end for
for all a_{rs} **do**
  if $\tau_{rs} < \tau_{min}$ **then**
    $\tau_{rs} = \tau_{min}$
  end if
  if $\tau_{rs} > \tau_{max}$ **then**
    $\tau_{rs} = \tau_{max}$
  end if
end for

Apéndice B

Calibración de las Metaheurísticas

Desde siempre en la utilización de metaheurísticas para resolver los problemas de optimización, los investigadores se han encontrado que una parte muy importante de sus investigaciones y la que determina en buena medida el buen desempeño de una metaheurística es el de encontrar el mejor seteo de valores de los parámetros que controlan los algoritmos. A pesar de la importancia del problema, poco se ha podido avanzar en este tema. Ciertamente existen guías que pretenden ayudar al investigador a encontrar los mejores valores. Pero todavía hoy en día, la mejor o peor calibración de los parámetros queda basado en la intuición y experiencia del investigador, basándose en una metodología de ensayo y error. Sin embargo, en los últimos años, se están empezando a utilizar herramientas de simulación en la calibración de las metaheurísticas, uno de ellas es la utilización de análisis factorial [LK91]. En este trabajo se dedicó un tiempo en analizar y estudiar la posibilidad de aplicar análisis factorial en la calibración de las metaheurísticas, pero se encontró que su aplicación quedaba fuera del alcance del mismo y que ameritaba un estudio mas profundo para poder aplicarlo con éxito. Como consecuencia, el método de calibración se basa en la observación del comportamiento de los diferentes algoritmos al variar los valores de los parámetros.

B.1. Método de Calibración

Tomando como base lo descrito anteriormente se define el siguiente método de calibración de las metaheurísticas implementadas en este trabajo:

- Paso 0: Separar los parámetros que controlan la metaheurística en cualitativos y cuantitativos. Los cuantitativos generalmente toman valores numéricos y los cualitativos (que controlan el comportamiento general de la metaheurística), generalmente ya se tiene determinado por la metaheurística implementada los

valores posibles que pueden tomar. Los valores que pueden tomar los parámetros cualitativos es infinito, mientras que los cuantitativos tienen un conjunto finito de valores posibles.

- Paso 1: Para los parámetros numéricos, mediante experimentación previa, se determinan los valores posibles y razonables que pueden tomar.

- Paso 2: A cada uno de esos parámetros se les fija uno de los valores posibles. También se fijan los valores para los parámetros cualitativos.

- Paso 3: Del paso 1 y paso 2, se tiene para cada parámetro numérico un valor fijo y un conjunto de valores posibles y para cada parámetro cualitativo un valor fijo. Para los problemas 20032-1 y 20041-1 se realizan los experimentos variando cada uno de los parámetros y dejando el resto fijo. Cada experimento se ejecuta 3 veces durante 10 minutos.

- Paso 4: Del resultados del paso 3 se elige para cada parámetro numérico los mejores 5 valores y se fija de nuevo los valores fijos de cada uno. Se fijan los valores iniciales para los parámetros cualitativos. Se toman los problemas 20032-5 y 20041-5 y el conjunto completo de parámetros y se realizan 5 corridas de 30', variando cada uno de los parámetros y dejando el resto fijo.

En cada una de las corridas para los pasos 3 y 4 se utilizan las mismas soluciones iniciales y lo único que varía es la semilla para la generación de los números aleatorios.

B.2. Soluciones iniciales

En la sección 4.10 se describe como se construyen las soluciones iniciales, tanto las aleatorias como las construidas. Para la calibración de las metaheurísticas, previamente, se construyen 100 soluciones aleatorias y 100 soluciones construidas para cada problema utilizado. GA utiliza las primeras soluciones según el tamaño de la población definida y TS toma como solución inicial la primera generada. Los algoritmos ACO no utilizan las soluciones iniciales ya que cada hormiga construye su solución a partir de una solución vacía.

B.3. Calibración de Metaheurísticas

Utilizando el método definido anteriormente se realiza la calibración de las metaheurísticas adaptadas al PASHA.

B.3.1. Algoritmos Genéticos

Los parámetros que controlan GA aplicado al PASHA son:

- Tipo de Población Inicial:
  - A - Población inicial aleatoria,
  - C - Población inicial construida por un algoritmo goloso (greedy).
- Tipo de Reproducción:
  - P - Reproducción Proporcional a la aptitud,
  - R - Reproducción por Ranking,
  - T - Reproducción por Torneo.
- Tamaño de la Población: N^+
- Cantidad de Puntos de Cruzamiento: N^+
- Probabilidad de Cruzamiento: $[0, 1]$
- Probabilidad de Mutación: $[0, 1]$
- Probabilidad de Torneo: $[0, 1]$

Paso 0

Se separan los parámetros en cualitativos y cuantitativos:

- Cualitativos:
  - Tipo de Población Inicial
  - Tipo de Reproducción
- Cuantitativos:
  - Tamaño de la Población
  - Cantidad de Puntos de Cruzamiento
  - Probabilidad de Cruzamiento
  - Probabilidad de Mutación
  - Probabilidad de Torneo

Paso 1

Se realizan algunas pruebas de ejecución del algoritmo para los problemas reales tomando diferentes valores de los parámetros y se obtienen las siguientes precisiones sobre los parámetros:

- Tamaño de la Población, en poblaciones más grandes (mayor a 100) el algoritmo necesita mas tiempo para realizar las operaciones además de consumir mas recurso de memoria, por lo que es preferible manejar poblaciones menores a 100 individuos.
- Cantidad de Puntos de Cruzamiento, no se obtienen diferencias significativas, solo que la cantidad de puntos no puede ser mayor al largo del cromosoma.
- Probabilidad de Cruzamiento, cuanto mayor sea el material genético cruzado (mayor a 0.5) se obtienen mejores resultados.
- Probabilidad de Mutación, debe ser pequeña, menor que 0.05.
- Probabilidad de Torneo, menor que 0,5.

Paso 2

De las pruebas realizadas en el paso anterior se fijan los diferentes valores posibles de los parámetros:

- Tamaño de la Población = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100}
- Puntos de Cruzamiento = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100}
- Probabilidad de Cruzamiento = {0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 1.0}
- Probabilidad de Mutación = {0, 0.005, 0.01, 0.015, 0.02, 0.025, 0.03, 0.035, 0.04, 0.045, 0.05}
- Probabilidad de Torneo = {0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5}

Y se fija cada parámetro en:

- Tipo de Población Inicial = C
- Tipo de Reproducción = R

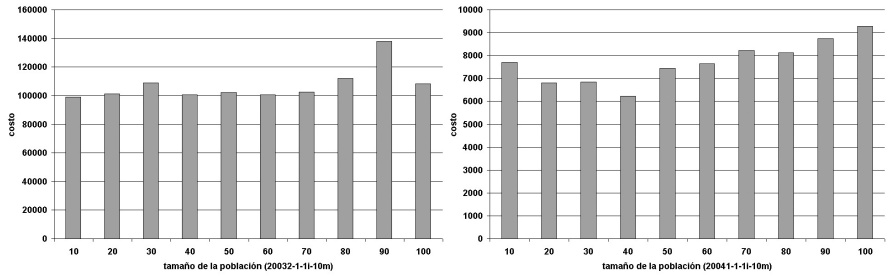
- Tamaño de la Población = 50
- Puntos de Cruzamiento = 50
- Probabilidad de Cruzamiento = 0.75
- Probabilidad de Mutación = 0.025
- Probabilidad de Torneo = 0.25

Paso 3

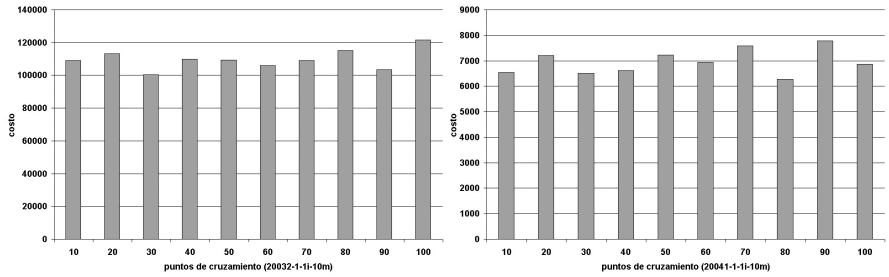
Con los valores de los parámetros definidos en el paso anterior, se varían los valores de cada uno de los parámetros (Tamaño de la Población, Puntos de Cruzamiento, Probabilidad de Cruzamiento, Probabilidad de Mutación, Probabilidad de Torneo) por vez, dejando el resto de ellos fijos, para los problemas 20032-1 y 20041-1 y se realiza una corrida de 10 minutos ($2*(10+10+11+11+11)*10' = 1060' = 17hs40'$).

Los resultados que se obtienen se representan en las siguientes gráficas:

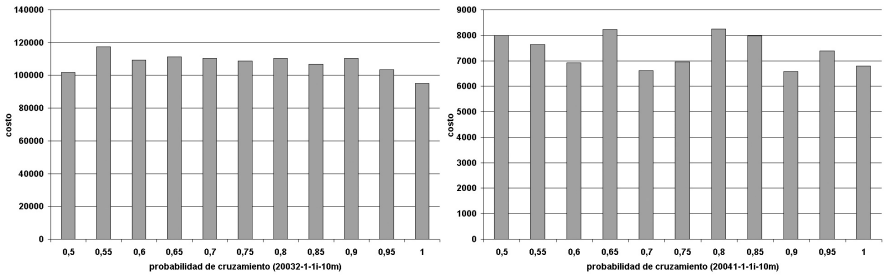
- Tamaño de la Población



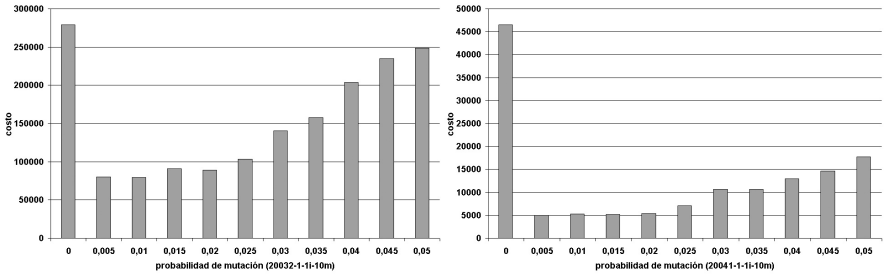
- Puntos de Cruzamiento



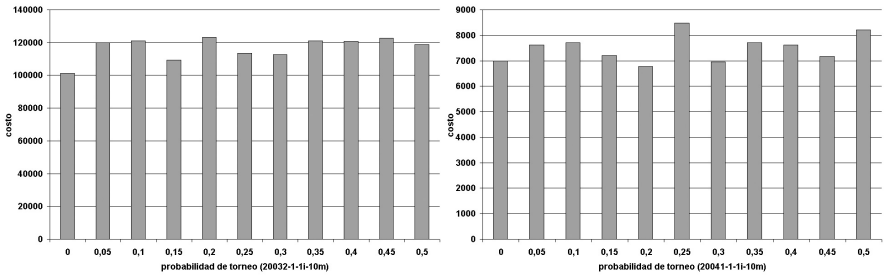
- Probabilidad de Cruzamiento



- Probabilidad de Mutación



- Probabilidad de Torneo



Se eligen los 5 mejores valores de cada parámetro:

- Tipo de Población Inicial = {A, C}
- Tipo de Reproducción = {P, R, T}
- Tamaño de la Población = {20, 40, 60, 80, 100}
- Puntos de Cruzamiento = {10, 30, 50, 70, 90}
- Probabilidad de Cruzamiento = {0.6, 0.7, 0.8, 0.9, 1.0}
- Probabilidad de Mutación = {0.005, 0.01, 0.015, 0.02, 0.025}

- Probabilidad de Torneo = {0.1, 0.2, 0.3, 0.4, 0.5}

Y se fija cada parámetro en:

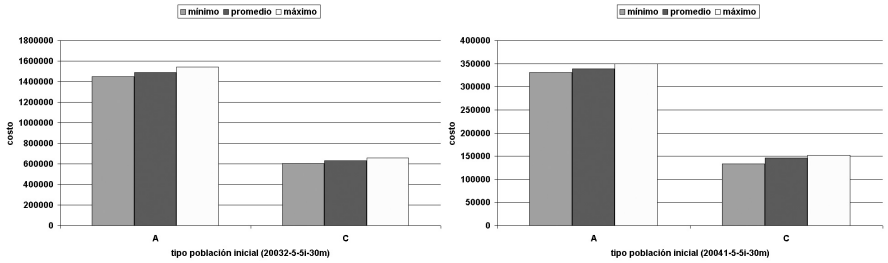
- Tipo Población Inicial = C
- Tipo de Reproducción = R
- Tamaño de la Población = 60
- Puntos de Cruzamiento = 50
- Probabilidad de Cruzamiento = 0.9
- Probabilidad de Mutación = 0.005
- Probabilidad de Torneo = 0.3

Paso 4

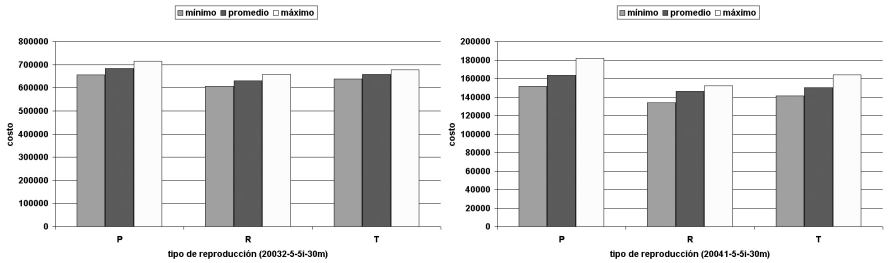
Con los valores de los parámetros fijados en el paso anterior, se toman los problemas reales completos (20032-5 y 20041-5) y se realizan 5 ejecuciones de 30' c/u ($2 * (2 + 3 + 5 + 5 + 5 + 5 + 5) * 5 * 30' = 9000' = 150hs$).

Los resultados que se obtienen se representan en las siguientes gráficas:

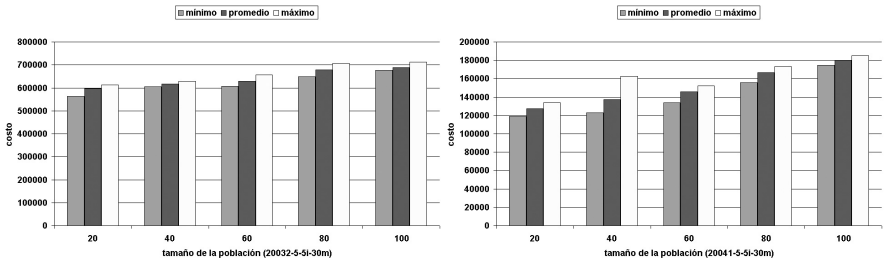
- Tipo Población Inicial



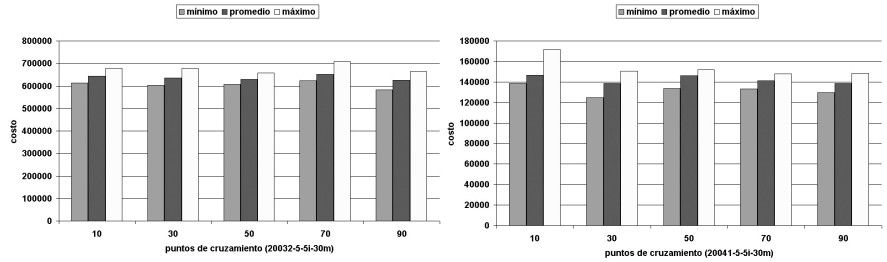
- Tipo de Reproducción



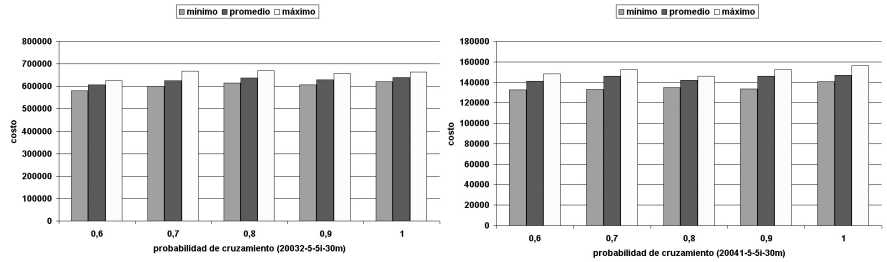
- Tamaño de la Población



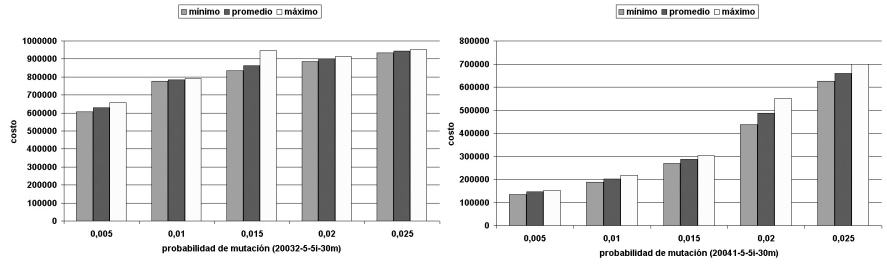
- Puntos de Cruzamiento



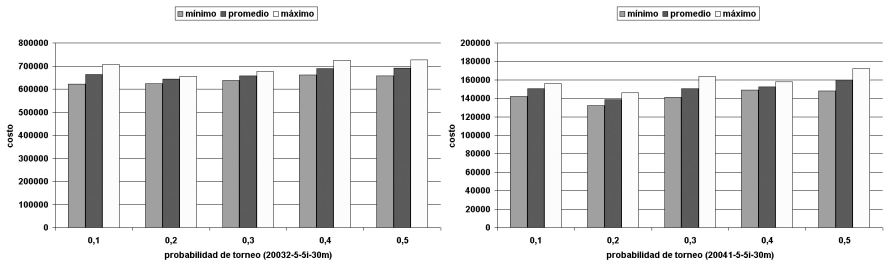
- Probabilidad de Cruzamiento



- Probabilidad de Mutación



- Probabilidad de Torneo



En el siguiente cuadro se muestra, para cada parámetro, el valor con el cual se obtiene el menor costo, independientemente del valor de los otros parámetros, considerando todas las corridas realizadas en el Paso 4:

Parámetros	Problemas 20032-5	20041-5
Tipo de Población Inicial	C	C
Tipo de Reproducción	R	T
Tamaño de la Población	20	20
Puntos de Cruzamiento	90	20
Probabilidad de Cruzamiento	0.6	0.6
Probabilidad de Mutación	0.005	0.005
Probabilidad de Torneo		0.2

y en el cuadro siguiente se muestran los valores de cada parámetro con los cuales se obtiene el mejor resultado para cada uno de los problemas. Y el valor de los mismos con los cuales se obtiene el mejor promedio, considerando las 5 ejecuciones que se realizan para cada valor de los parámetros:

Parámetros	Problemas 20032-5 mejor	20032-5 promedio	20041-5 mejor	20041-5 promedio
Tipo de Población Inicial	C	C	C	C
Tipo de Reproducción	R	R	R	R
Tamaño de la Población	20	20	20	20
Puntos de Cruzamiento	50	50	50	50
Probabilidad de Cruzamiento	0.9	0.9	0.9	0.9
Probabilidad de Mutación	0.005	0.005	0.005	0.005
Probabilidad de Torneo				
	563762	597403.20	118992	127478.6

B.3.2. Búsqueda Tabú

Los parámetros que controlan TS aplicado al PASHA son:

- Tipo de Solución Inicial:
  - A - Solución inicial aleatoria,
  - C - Solución inicial construida por un algoritmo goloso (greedy).
- Cantidad de Soluciones Vecinas: N^+
- Largo Lista Tabú Clase: N^+
- Largo Lista Tabú Salón Horario: N^+

Paso 0

Se separan los parámetros en cualitativos y cuantitativos:

- Cualitativos:
  - Tipo de Solución Inicial
- Cuantitativos:
  - Cantidad de Soluciones Vecinas
  - Largo Lista Tabú Clase
  - Largo Lista Tabú Salón Horario

Paso 2

Se fijan los valores posibles de los diferentes parámetros:

- Cantidad de Soluciones Vecinas = {1, 2, 5, 10, 15, 20, 30, 50, 100, 150, 200, 300, 500, 800, 1000, 1500, 2000, 5000, 10000, 20000}
- Largo Lista Tabú Clase = {1, 2, 5, 10, 15, 20, 30, 50, 100, 150, 200, 300, 500, 800, 1000, 1500, 2000, 5000, 10000, 20000}
- Largo Lista Tabú Salón Horario = {1, 2, 5, 10, 15, 20, 30, 50, 100, 150, 200, 300, 500, 800, 1000, 1500, 2000, 5000, 10000, 20000}

Y se fija cada parámetro en:

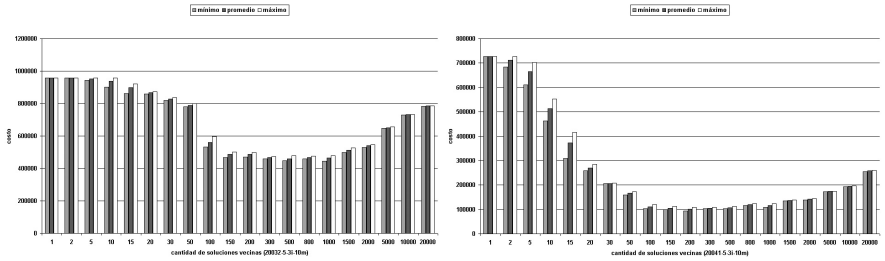
- Tipo de Solución Inicial = C
- Cantidad de Soluciones Vecinas = 150
- Largo Lista Tabú Clase = 150
- Largo Lista Tabú Salón Horario = 150

Paso 3

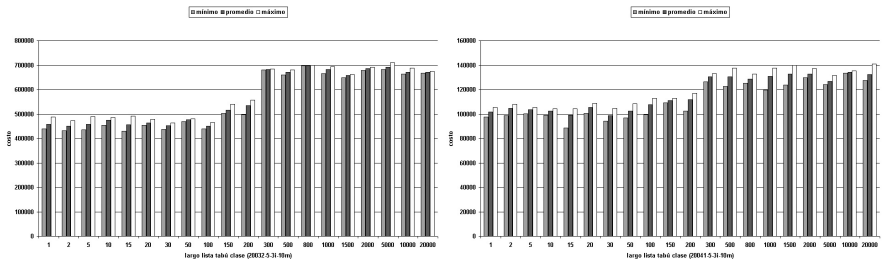
Con los valores de los parámetros definidos en el paso anterior, se varían los valores de cada uno de los parámetros (Cantidad de Soluciones Vecinas, Largo Lista Tabú Clase, Largo Lista Tabú Salón Horario) por vez, dejando el resto de ellos fijos, para los problemas 20032-5 y 20041-5 y se realizan 3 ejecuciones de 10 minutos ($2 * (20 + 20 + 20) * 3 * 10' = 3600' = 60hs$).

Los resultados que se obtienen se representan en las siguientes gráficas:

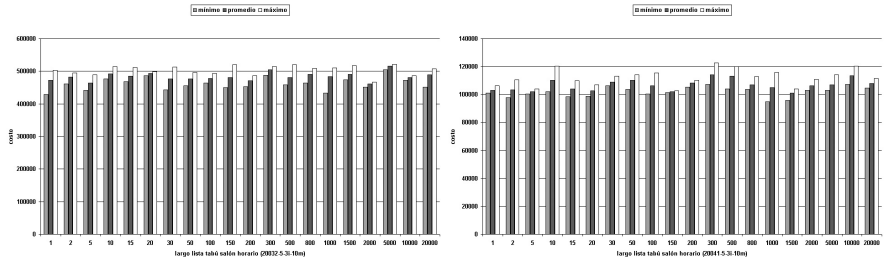
- Cantidad de Soluciones Vecinas



- Largo Lista Tabú Clase



- Largo Lista Tabú Salón Horario



Se eligen los 5 mejores valores de cada parámetro:

- Tipo de Solución Inicial = {A, C}
- Cantidad de Soluciones Vecinas = {100, 200, 500, 800, 1000}
- Largo Lista Tabú Clase = {1, 10, 15, 30, 100}
- Largo Lista Tabú Salón Horario = {1, 5, 30, 200, 1500}

Y se fija cada parámetro en:

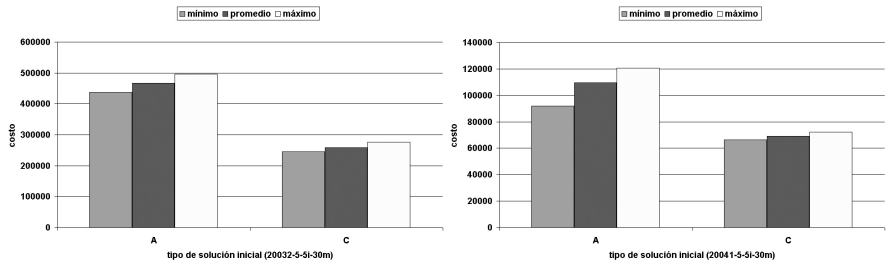
- Tipo de Solución Inicial = C
- Cantidad de Soluciones Vecinas = 500
- Largo Lista Tabú Clase = 30
- Largo Lista Tabú Salón Horario = 200

Paso 4

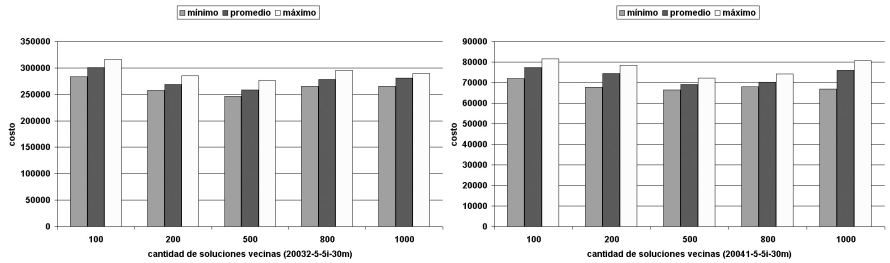
Con los valores de los parámetros fijados en el paso anterior, se toman los problemas reales completos (20032-5 y 20041-5) y se realizan 5 ejecuciones de 30' c/u ($2 * (2 + 5 + 5 + 5) * 5 * 30' = 5100' = 85hs$).

Los resultados que se obtienen se representan en las siguientes gráficas:

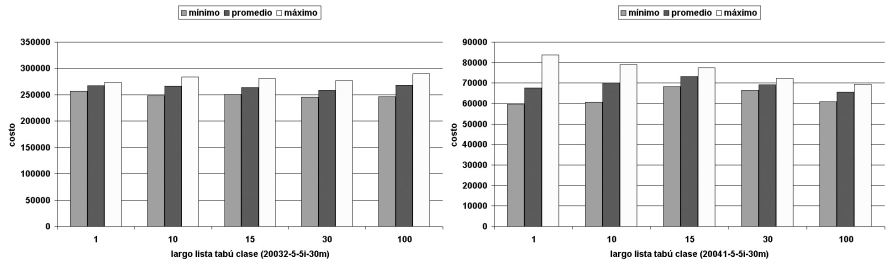
- Tipo de Solución Inicial



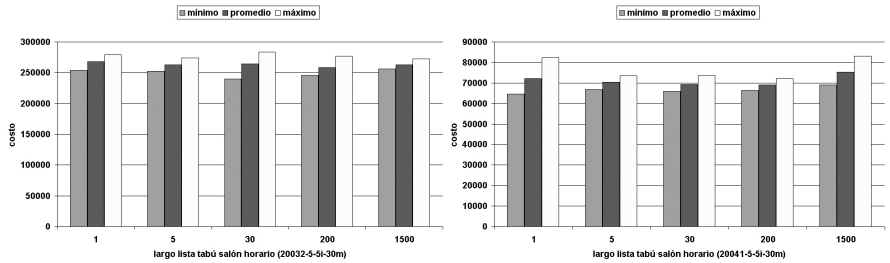
- Cantidad de Soluciones Vecinas



- Largo Lista Tabú Clase



- Largo Lista Tabú Salón Horario



En el siguiente cuadro se muestra, para cada parámetro, el valor con el cual se obtiene el menor costo, independientemente del valor de los otros parámetros, considerando todas las corridas realizadas en el Paso 4:

Parámetros	Problemas 20032-5	20041-5
Tipo de Solución Inicial	C	C
Cantidad de Soluciones Vecinas	500	500
Largo Lista Tabú Clase	30	100
Largo Lista Tabú Salón Horario	200	200

y en el cuadro siguiente se muestran los valores de cada parámetro con los cuales se obtiene el mejor resultado para cada uno de los problemas. Y el valor de los mismos con los cuales se obtiene el mejor promedio, considerando las 5 ejecuciones que se realizan para cada valor de los parámetros:

Parámetros	Problemas 20032-5 mejor	20032-5 promedio	20041-5 mejor	20041-5 promedio
Tipo de Solución Inicial	C	C	C	C
Cantidad de Soluciones Vecinas	500	500	500	500
Largo Lista Tabú Clase	30	30	1	100
Largo Lista Tabú Salón Horario	30	200	200	200
	240178	258416.20	59594	65486

B.3.3. Ant System

Los parámetros que controlan AS aplicado al PASHA son:

- Cantidad de Hormigas: N^+
- Rastro Inicial: R^+
- Evaporación: $[0, 1]$
- Ponderación del Rastro: R^+
- Ponderación de la Visibilidad: R^+

Paso 1

Se realizan algunas pruebas de ejecución del algoritmo para los problemas reales tomando diferentes valores de los parámetros y se obtienen las siguientes precisiones sobre los parámetros:

- Cantidad de Hormigas, no se encuentra diferencias significativas en la cantidad de hormigas utilizadas.
- Rastro Inicial, idem anterior.
- Evaporación, mejores resultados con la evaporación cercana a 0.
- Ponderación del Rastro: no se encuentran diferencias significativas.
- Ponderación de la Visibilidad, mejores resultados con la ponderación mayor a 1.

Paso 2

De las pruebas realizadas en el paso anterior se fijan los diferentes valores posibles de los parámetros:

- Cantidad de Hormigas = {1, 2, 5, 10, 15, 20, 25, 30, 40, 50}
- Rastro Inicial = {0.01, 0.1, 0.5, 1, 2, 5, 10, 20, 50, 100}
- Evaporación = {0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 0.3, 0.5, 0.9}
- Ponderación del Rastro = {0.01, 0.1, 0.5, 1, 2, 5, 10, 20, 50, 100}
- Ponderación de la Visibilidad = {0.5, 1, 2, 5, 10, 15, 20, 30, 50, 100}

Y se fija cada parámetro en:

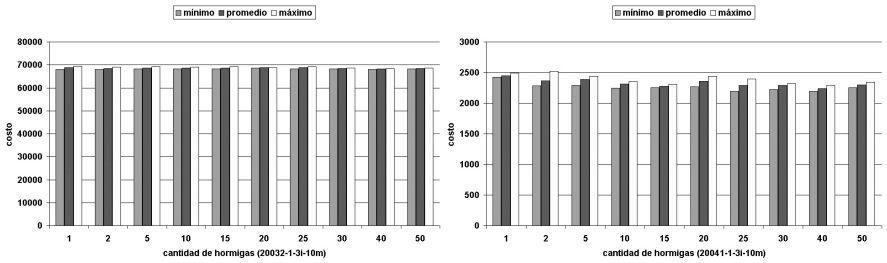
- Cantidad de Hormigas = 1
- Rastro Inicial = 0.01
- Evaporación = 0.01
- Ponderación del Rastro = 50
- Ponderación de la Visibilidad = 20

Paso 3

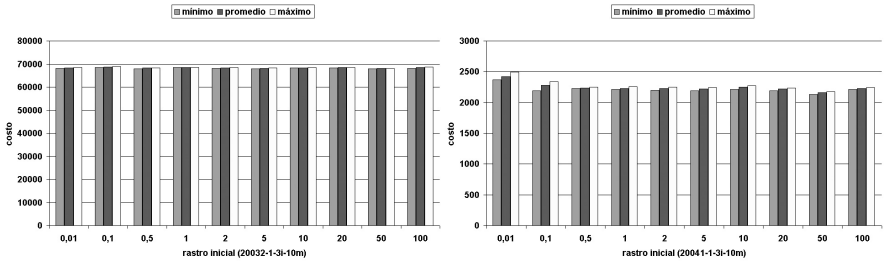
Con los valores de los parámetros definidos en el paso anterior, se varían los valores de cada uno de los parámetros, dejando el resto de ellos fijos, para los problemas 20032-5 y 20041-5 y se realizan 3 ejecuciones de 10 minutos ($2*(10+10+10+10+10)*3*10' = 3000' = 50hs$).

Los resultados que se obtienen se representan en las siguientes gráficas:

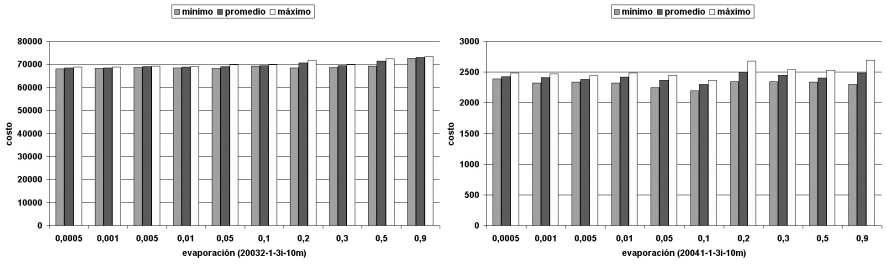
- Cantidad de Hormigas



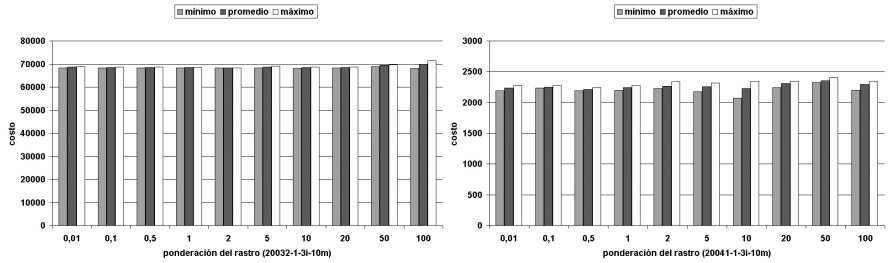
- Rastro Inicial



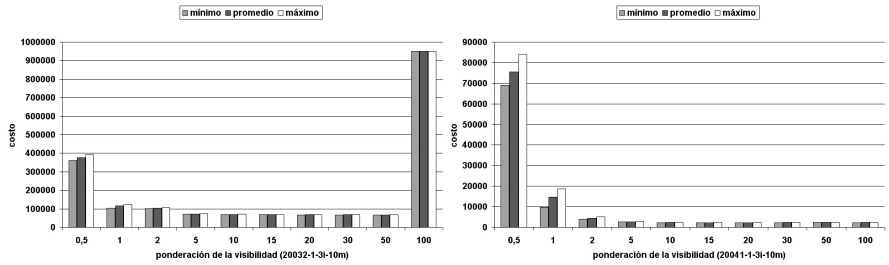
- Evaporación



- Ponderación del Rastro



- Ponderación de la Visibilidad



Se eligen los 5 mejores valores de cada parámetro:

- Cantidad de Hormigas = {2, 5, 10, 15, 40}
- Rastro Inicial = {1, 5, 10, 20, 50}
- Evaporación = {0.0005, 0.001, 0.005, 0.05, 0.1}
- Ponderación del Rastro = {0.01, 0.5, 1, 2, 10}
- Ponderación de la Visibilidad = {10, 15, 20, 30, 50}

Y se fija cada parámetro en:

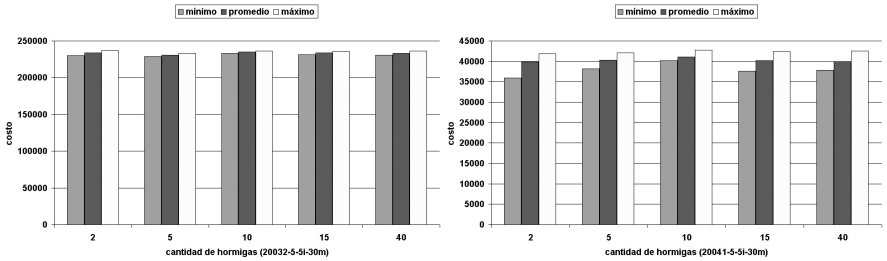
- Cantidad de Hormigas = 40
- Rastro Inicial = 50
- Evaporación = 0.0005
- Ponderación del Rastro = 10
- Ponderación de la Visibilidad = 20

Paso 4

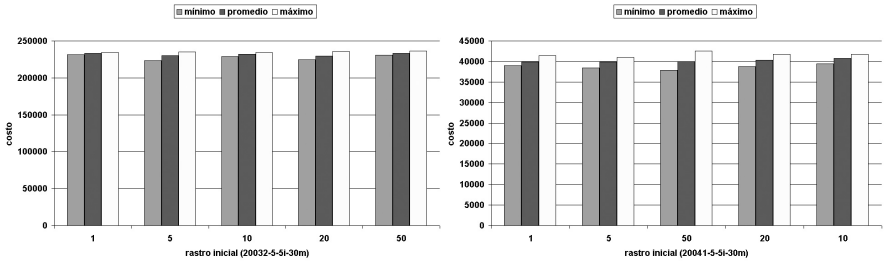
Con los valores de los parámetros fijados en el paso anterior, se toman los problemas reales completos (20032-5 y 20041-5) y se realizan 5 ejecuciones de 30' c/u ($2 * (5 + 5 + 5 + 5 + 5) * 5 * 30' = 7500' = 125hs$).

Los resultados que se obtienen se representan en las siguientes gráficas:

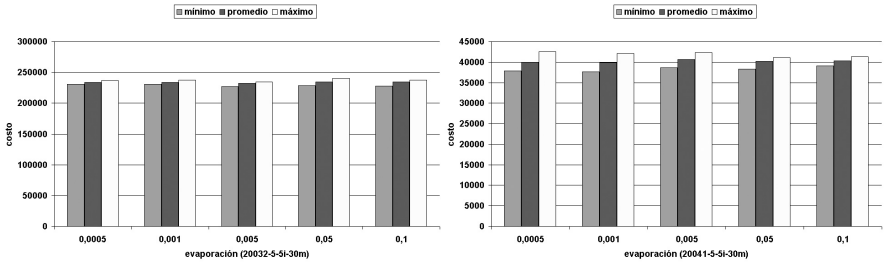
- Cantidad de Hormigas



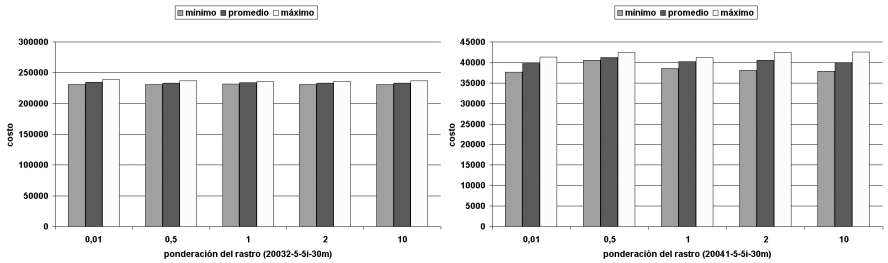
- Rastro Inicial



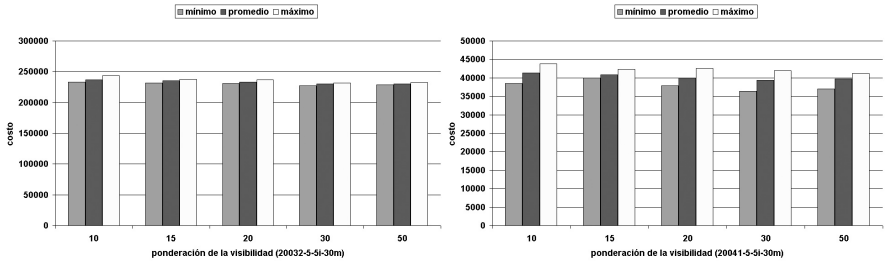
- Evaporación



- Ponderación del Rastro



- Ponderación de la Visibilidad



En el siguiente cuadro se muestra, para cada parámetro, el valor con el cual se obtiene el menor costo, independientemente del valor de los otros parámetros, considerando todas las corridas realizadas en el Paso 4:

Parámetros	Problemas 20032-5	20041-5
Cantidad de Hormigas	5	2
Rastro Inicial	20	1
Evaporación	0.005	0.001
Ponderación del Rastro	0.5	0.01
Ponderación de la Visibilidad	30	30

y en el cuadro siguiente se muestran los valores de cada parámetro con los cuales se obtiene el mejor resultado para cada uno de los problemas. Y el valor de los mismos con los cuales se obtiene el mejor promedio, considerando las 5 ejecuciones que se realizan para cada valor de los parámetros:

	Problemas			
	20032-5		20041-5	
Parámetros	mejor	promedio	mejor	promedio
Cantidad de Hormigas	40	40	2	40
Rastro Inicial	5	20	50	50
Evaporación	0.0005	0.0005	0.0005	0.0005
Ponderación del Rastro	10	10	10	10
Ponderación de la Visibilidad	20	20	20	30
	223544	229812.2	35913	39396.8

B.3.4. Ant Colony System

Los parámetros que controlan ACS aplicado al PASHA son:

- Cantidad de Hormigas: N^+
- Rastro Inicial: R^+
- Evaporación: $[0, 1]$
- Evaporación Local: $[0, 1]$
- Probabilidad q0: $[0, 1]$
- Ponderación de la Visibilidad: R^+

Paso 1

Se realizan algunas pruebas de ejecución del algoritmo para los problemas reales tomando diferentes valores de los parámetros y se obtienen las siguientes precisiones sobre los parámetros:

- Cantidad de Hormigas, cantidad de hormigas entre 20 y 40 parece ser los que dan mejor resultados, para cantidades mayores de 40 el algoritmo demora mucho tiempo en cada iteración que lo hace inutilizable en los tiempos manejados en la calibración.

- Rastro Inicial, no se encuentran diferencias significativas para diferentes valores del rastro inicial.
- Evaporación, mejores resultados con la evaporación cercana a 0, menor a 0.5.
- Evaporación Local, no se encuentran diferencias significativas para diferentes evaporaciones locales.
- Probabilidad q0, idem anterior.
- Ponderación de la Visibilidad, mejores resultados con la ponderación mayor a 0.5 y menor a 100.

Paso 2

De las pruebas realizadas en el paso anterior se fijan los diferentes valores posibles de los parámetros:

- Cantidad de Hormigas = {1, 2, 5, 10, 15, 20, 25, 30, 35, 40}
- Rastro Inicial = {0.01, 0.1, 0.5, 1, 2, 5, 10, 20, 50, 100}
- Evaporación = {0.0001, 0.0005, 0.005, 0.01, 0.05, 0,1, 0.2, 0.3, 0.4, 0.5}
- Evaporación Local = {0.0005, 0.001, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 0.9}
- Probabilidad q0 = {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95}
- Ponderación de la Visibilidad = {1, 2, 5, 10, 15, 20, 30, 40, 50, 100}

Y se fija cada parámetro en:

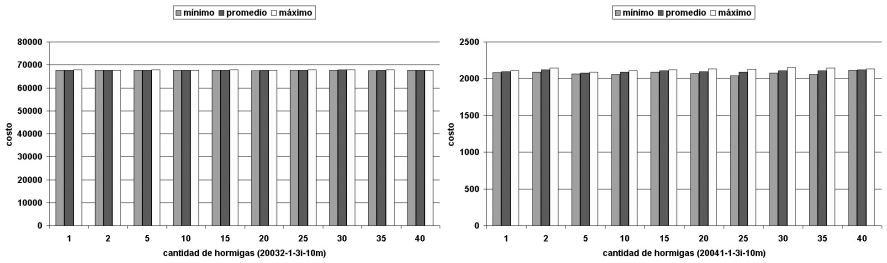
- Cantidad de Hormigas = 5
- Rastro Inicial = 10
- Evaporación = 0.01
- Evaporación Local = 0.2
- Probabilidad q0 = 0.5
- Ponderación de la Visibilidad = 20

Paso 3

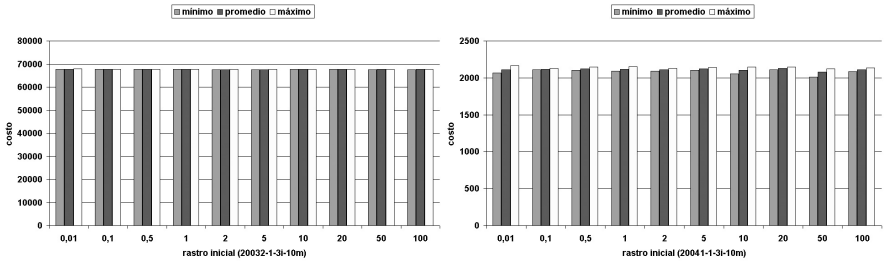
Con los valores de los parámetros definidos en el paso anterior, se varían los valores de cada uno de los parámetros, dejando el resto de ellos fijos, para los problemas 20032-5 y 20041-5 y se realizan 3 ejecuciones de 10 minutos ($2*(10+10+10+10+10+10)*3*10' = 3600' = 60hs$).

Los resultados que se obtienen se representan en las siguientes gráficas:

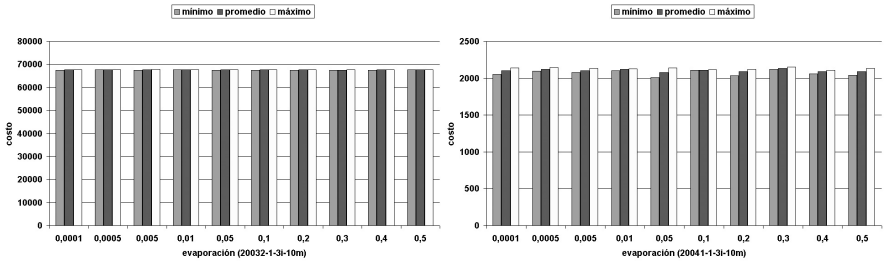
- Cantidad de Hormigas



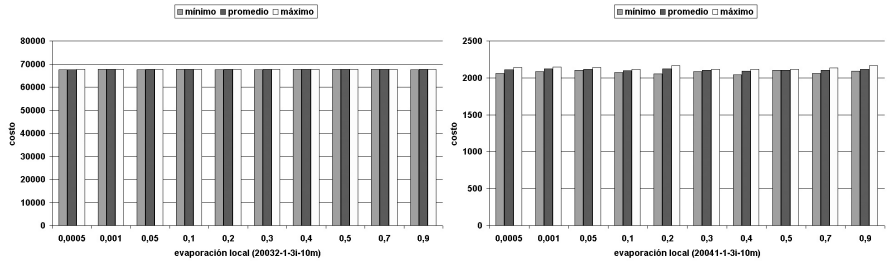
- Rastro Inicial



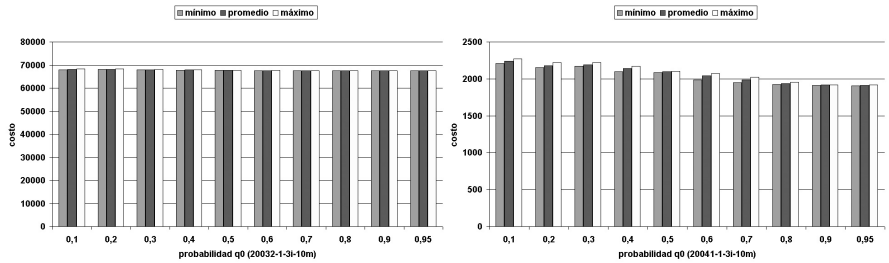
- Evaporación



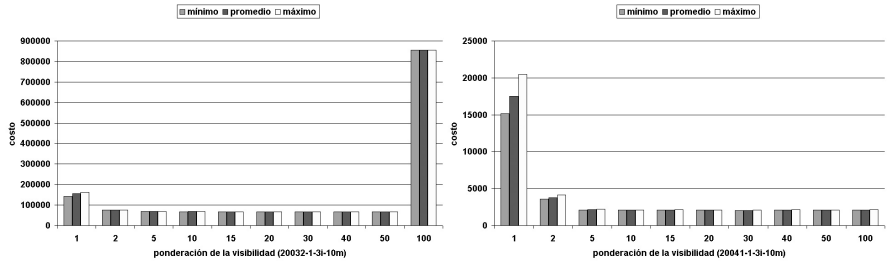
- Evaporación Local



- Probabilidad q0



- Ponderación de la Visibilidad



Se eligen los 5 mejores valores de cada parámetro:

- Cantidad de Hormigas = {2, 5, 10, 20, 30}
- Rastro Inicial = {0.01, 2, 10, 50, 100}
- Evaporación = {0.0001, 0.05, 0.1, 0.2, 0.4}
- Evaporación Local = {0.0005, 0.001, 0.1, 0.3, 0.7}
- Probabilidad q0 = {0.6, 0.7, 0.8, 0.9, 0.95}
- Ponderación de la Visibilidad = {10, 20, 30, 40, 50}

Y se fija cada parámetro en:

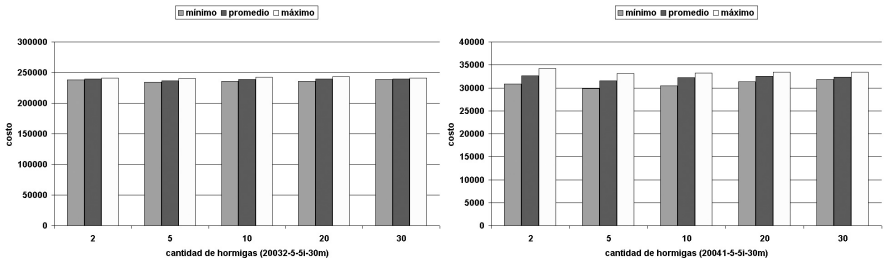
- Cantidad de Hormigas = 10
- Rastro Inicial = 50
- Evaporación = 0.05
- Evaporación Local = 0.1
- Probabilidad q0 = 0.9
- Ponderación de la Visibilidad = 30

Paso 4

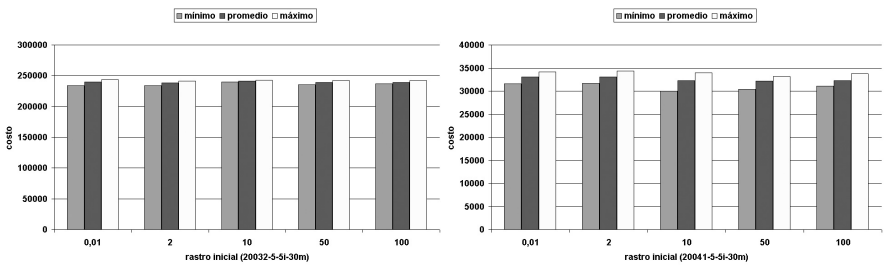
Con los valores de los parámetros fijados en el paso anterior, se toman los problemas reales completos (20032-5 y 20041-5) y se realizan 5 ejecuciones de 30' c/u ($2 * (5 + 5 + 5 + 5 + 5) * 5 * 30' = 9000' = 150hs$).

Los resultados que se obtienen se representan en las siguientes gráficas:

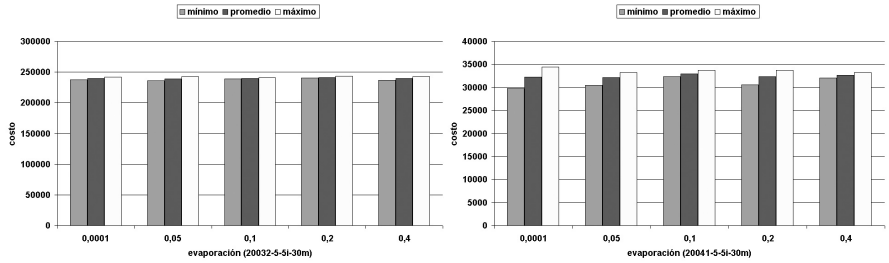
- Cantidad de Hormigas



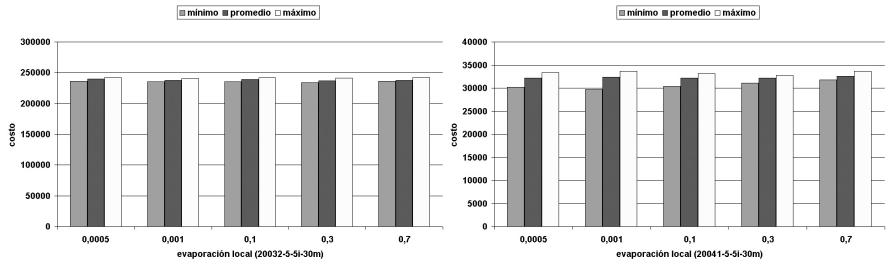
- Rastro Inicial



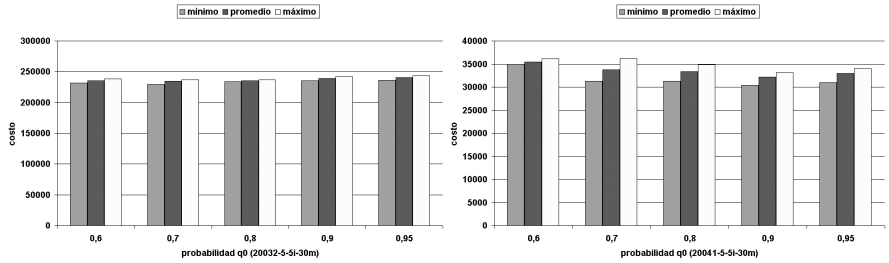
- Evaporación



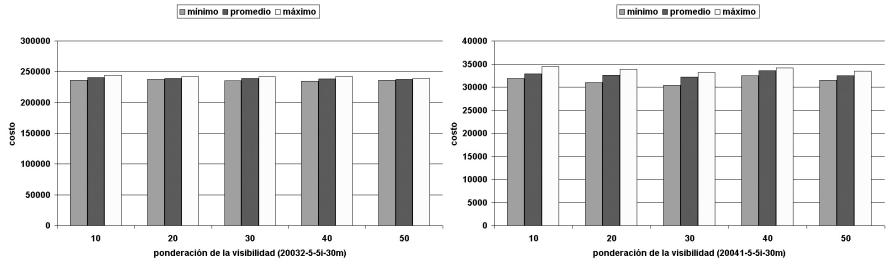
- Evaporación Local



- Probabilidad q0



- Ponderación de la Visibilidad



En el siguiente cuadro se muestra, para cada parámetro, el valor con el cual se obtiene el menor costo, independientemente del valor de los otros parámetros, considerando todas las corridas realizadas en el Paso 4:

Parámetros	Problemas 20032-5	20041-5
Cantidad de Hormigas	5	5
Rastro Inicial	2	50
Evaporación	0.05	0.05
Evaporación Local	0.3	0.0005
Probabilidad q0	0.7	0.9
Ponderación de la Visibilidad	50	30

y en el cuadro siguiente se muestran los valores de cada parámetro con los cuales se obtiene el mejor resultado para cada uno de los problemas. Y el valor de los mismos con los cuales se obtiene el mejor promedio, considerando las 5 ejecuciones que se realizan para cada valor de los parámetros:

Parámetros	Problemas 20032-5 mejor	20032-5 promedio	20041-5 mejor	20041-5 promedio
Cantidad de Hormigas	10	10	10	5
Rastro Inicial	50	50	50	50
Evaporación	0.05	0.05	0.05	0.05
Evaporación Local	0.1	0.1	0.001	0.1
Probabilidad q0	0.7	0.7	0.9	0.9
Ponderación de la Visibilidad	30	30	30	30
	229649	234325.60	29664	31543.40

B.3.5. $\mathcal{MAX} - \mathcal{MIN}$ Ant System

Los parámetros que controlan $\mathcal{MM}$AS aplicado al PASHA son:

- Cantidad de Hormigas: N^+
- Rastro Mínimo: R^+
- Rastro Máximo: R^+
- Evaporación: $[0, 1]$
- Ponderación del Rastro: R^+
- Ponderación de la Visibilidad: R^+

Paso 1

Se realizan algunas pruebas de ejecución del algoritmo para los problemas reales tomando diferentes valores de los parámetros y se obtienen las siguientes precisiones sobre los parámetros:

- Cantidad de Hormigas, cantidad de hormigas entre 20 y 40 parece ser los que dan mejor resultados, para cantidades mayores de 40 el algoritmo demora mucho tiempo en cada iteración que lo hace inutilizable en los tiempos manejados en la calibración.
- Rastro Mínimo, no se encuentran diferencias significativas para los diferentes valores del rastro mínimo.
- Rastro Máximo, idem anterior.
- Evaporación, idem anterior.
- Ponderación del Rastro, mejores resultados con la ponderación menor a 100.
- Ponderación de la Visibilidad, idem anterior.

Paso 2

De las pruebas realizadas en el paso anterior se fijan los diferentes valores posibles de los parámetros:

- Cantidad de Hormigas = {1, 2, 5, 10, 15, 20, 25, 30, 35, 40}
- Rastro Mínimo = {0.01, 0.2, 1, 10, 20, 30, 50, 100, 1000, 2000}
- Rastro Máximo = {0.01, 0.2, 1, 10, 20, 30, 50, 100, 1000, 2000}
- Evaporación = {0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 0.3, 0.5, 0.9}
- Ponderación del Rastro = {0.01, 0.1, 0.5, 1, 2, 5, 10, 20, 50, 100}
- Ponderación de la Visibilidad = {0.5, 1, 2, 5, 10, 15, 20, 30, 50, 100}

Y se fija cada parámetro en:

- Cantidad de Hormigas = 10
- Rastro Mínimo = 0.01

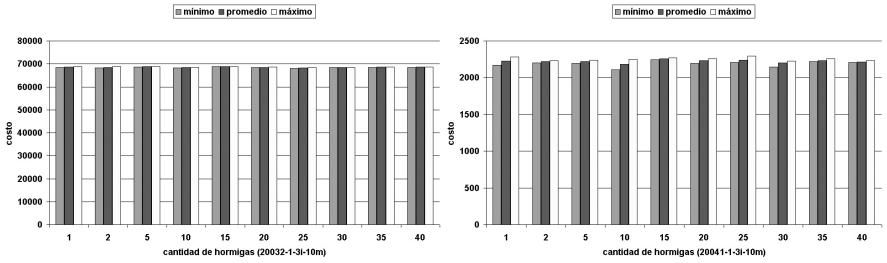
- Rastro Máximo = 2000
- Evaporación = 0.01
- Ponderación del Rastro = 10
- Ponderación de la Visibilidad = 20

Paso 3

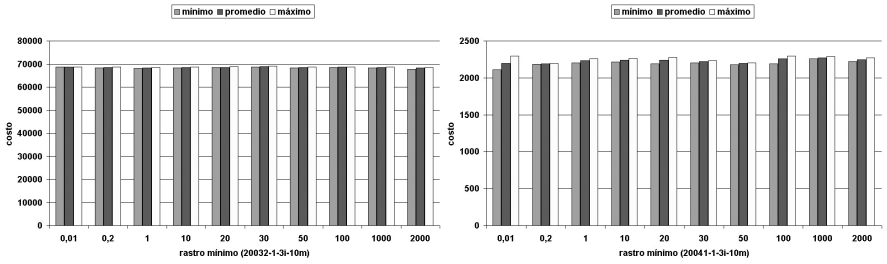
Con los valores de los parámetros definidos en el paso anterior, se varían los valores de cada uno de los parámetros, dejando el resto de ellos fijos, para los problemas 20032-1 y 20041-1 y se realizan 3 ejecuciones de 10 minutos ($2*(10+10+10+10+10+10)*3*10' = 3600' = 60hs$).

Los resultados que se obtienen se representan en las siguientes gráficas:

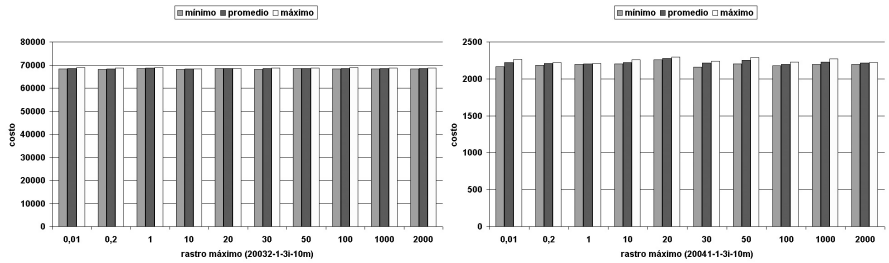
- Cantidad de Hormigas



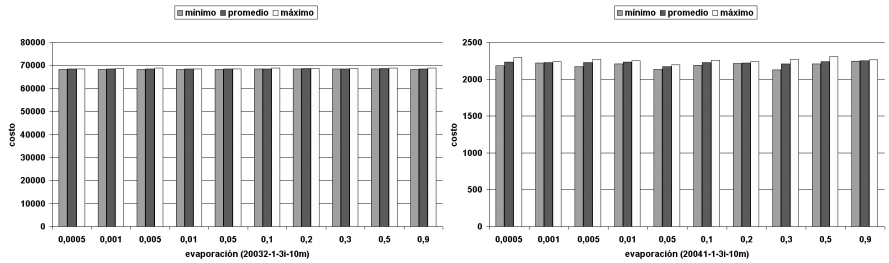
- Rastro Mínimo



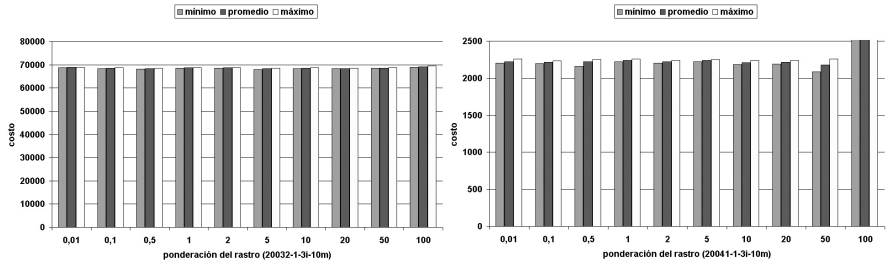
- Rastro Máximo



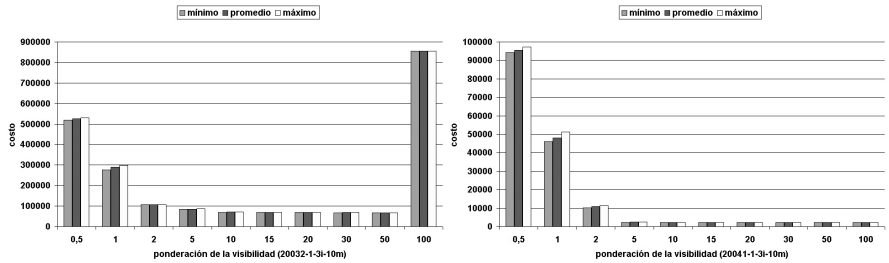
- Evaporación



- Ponderación del Rastro



- Ponderación de la Visibilidad



Se eligen los 5 mejores valores de cada parámetro:

- Cantidad de Hormigas = {2, 10, 20, 30, 40}
- Rastro Mínimo = {0.2, 1, 10, 20, 50}
- Rastro Máximo = {0.2, 1, 10, 30, 100}
- Evaporación = {0.0005, 0.001, 0.01, 0.05, 0.1}
- Ponderación del Rastro = {0.1, 0.5, 5, 10, 20}
- Ponderación de la Visibilidad = {10, 15, 20, 30, 50}

Y se fija cada parámetro en:

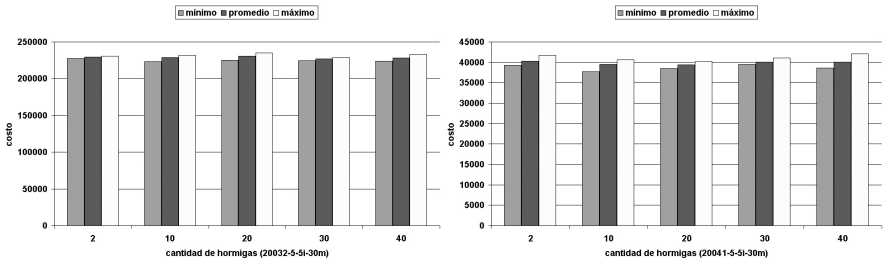
- Cantidad de Hormigas = 10
- Rastro Mínimo = 0.2
- Rastro Máximo = 100
- Evaporación = 0.01
- Ponderación del Rastro = 0.5
- Ponderación de la Visibilidad = 50

Paso 4

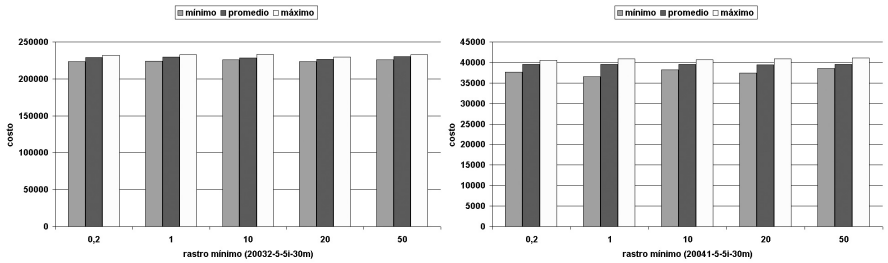
Con los valores de los parámetros fijados en el paso anterior, se toman los problemas reales completos (20032-5 y 20041-5) y se realizan 5 ejecuciones de 30' c/u ($2 * (5 + 5 + 5 + 5 + 5 + 5) * 5 * 30' = 9000' = 150hs$).

Los resultados que se obtienen se representan en las siguientes gráficas:

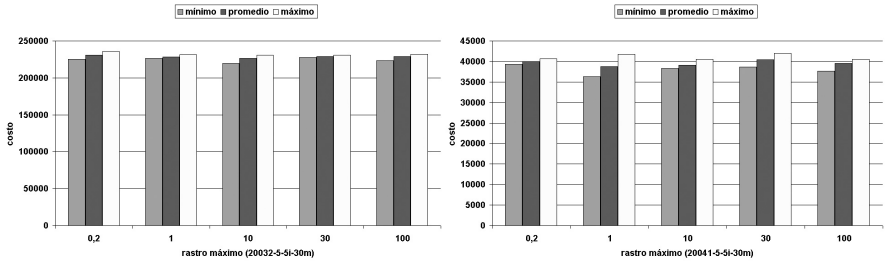
- Cantidad de Hormigas



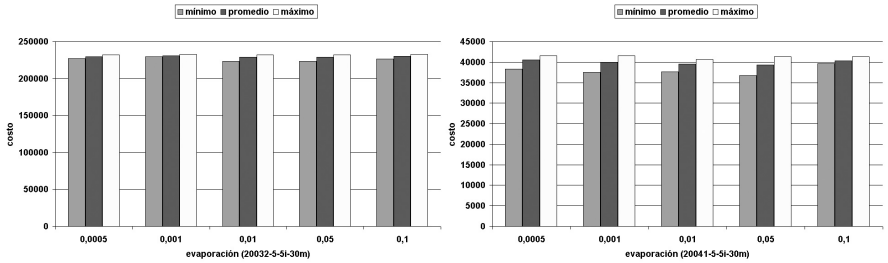
- Rastro Mínimo



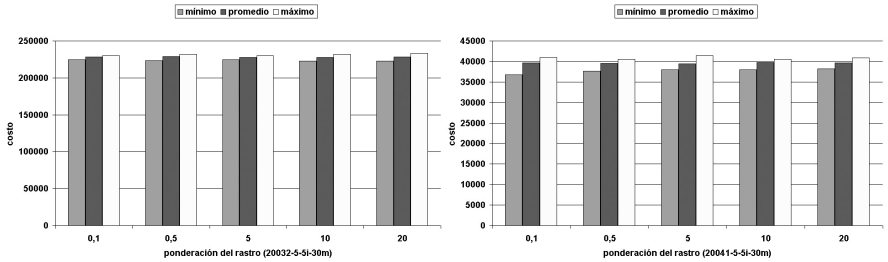
- Rastro Máximo



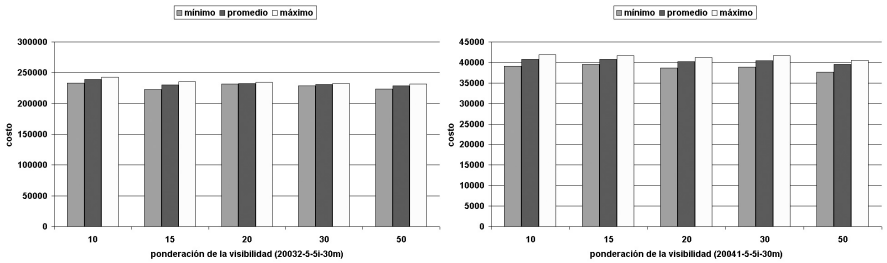
- Evaporación



- Ponderación del Rastro



- Ponderación de la Visibilidad



En el siguiente cuadro se muestra, para cada parámetro, el valor con el cual se obtiene el menor costo, independientemente del valor de los otros parámetros, considerando todas las corridas realizadas en el Paso 4:

Parámetros	Problemas 20032-5	20041-5
Cantidad de Hormigas	30	20
Rastro Mínimo	20	20
Rastro Máximo	100	100
Evaporación	0.01	0.05
Ponderación del Rastro	10	5
Ponderación de la Visibilidad	50	50

y en el cuadro siguiente se muestran los valores de cada parámetro con los cuales se obtiene el mejor resultado para cada uno de los problemas. Y el valor de los mismos con los cuales se obtiene el mejor promedio, considerando las 5 ejecuciones que se realizan para cada valor de los parámetros:

Parámetros	Problemas 20032-5 mejor	20032-5 promedio	20041-5 mejor	20041-5 promedio
Cantidad de Hormigas	10	10	10	10
Rastro Mínimo	0.2	0.2	0.2	0.2
Rastro Máximo	10	10	1	1
Evaporación	0.01	0.01	0.01	0.01
Ponderación del Rastro	0.5	0.5	0.5	0.5
Ponderación de la Visibilidad	50	50	50	50
	219444	226214.20	36278	38769

Bibliografía

[Abr91] D. Abramson. Constructing School Timetables Using Simulated Annealing: Sequential and Parallel Algorithms. *Management Science*, 37:98–113, 1991.

[ACT88] B. Angéniol, G. De La Croix, and J.Y. Le Texier. Self-Organizing Feature Maps and the Travelling Salesman Problem. *Neural Networks*, 1:289–293, 1988.

[ADK99] D.A. Abramson, H. Dang, and M. Krisnamoorthy. Simulated Annealing Cooling Schedules for the School Timetabling Problem. *Asia-Pacific Journal of Operational Research*, 16:1–12, 1999.

[AF89] J. Aubin and J.A. Ferland. A Large Scale Timetabling. *Computers and Operational Research*, 16(1):67–77, 1989.

[AK89a] H.L. Aarts and J.H.M Korst. Boltzmann Machine for Travelling Salesman Problems. *European Journal of Operational Research*, 39:79–95, 1989.

[AK89b] H.L. Aarts and J.H.M Korst. *Simulated Annealing and Boltzmann Machines*. John Wiley & Sons, Mew York, 1989.

[Akk73] E.A. Akkoyunlu. A Linear Algorithm for Computing the Optimum University Timetable. *Computer Journal*, 16:347–350, 1973.

[AM00] S. Abdennadher and M. Marte. University Course Timetabling using Constraint Handling Rules. *Journal of Applied Artificial Intelligence*, 14(4):311–326, 2000.

[AVCT02] R. Alvarez-Valdes, E. Crespo, and J.M. Tamarit. Design and Implementation of a Course Scheduling System Using Tabu Search. *European Journal of Operational Research*, 137:512–523, 2002.

[Bak85] J.E. Baker. Adaptative Selection Methods for Genetic Algorithms. In J.J. Grefenstette, editor, *First International Conference on Genetic Algorithms and their Applications*, pages 101–111. San Mateo, Morgan Kaufmann, 1985.

[Bal69] E. Balas. Machine Sequencing Via Disjunctive Graphs: An Implicit Enumeration Algorithm. *Journal of the Operational Research Society*, 17:941–957, 1969.

[Bar96] V.A. Bardadym. Computer-Aided School and University Timetabling: The New Wave. In E. Burke and P. Roos, editors, *The Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference*, volume 1153 of *Lecture Notes in Computer Science*, pages 22–45. Springer-Verlag, Berlin, Germany, 1996.

[BF97] R. Barber and G. Forster. The Syllabus Plus Exam Scheduler: Design Overview and Initial Results. In E. Burke and M. Carter, editors, *The Practice and Theory of Automated Timetabling: Selected Papers from the 2nd International Conference*, volume 1408 of *Lecture Notes in Computer Science*, pages 72–80. Springer-Verlag, Berlin, Germany, 1997.

[BGJ94] P. Boizumault, C. Gu'eret, and N. Jussien. Efficient Labelling and Constraint Relaxation for Solving Timetabling Problems. In *Workshop on Constraint Languages and their use in Problem Modelling, International Logic Programming Symposium*, pages 116–130. Ithaca, New York, 1994.

[BKJW97] E. Burke, J. Kingston, K. Jackson, and R. Weare. Automated University Timetabling: The State of the Art. *The Computer Journal*, 40(9):565–571, 1997.

[BLW92] N. Balakrishnan, A. Lucena, and R.T. Wong. Scheduling Examinations to Reduce Second-Order Conflicts. *Computers and Operational Research*, 19:353–361, 1992.

[BM85] J.R. Baker and G.B. McMahon. Scheduling the General Job-Shop. *Management Science*, 31(5):594–598, 1985.

[BNW96] E. Burke, J.Ñewall, and R. Weare. A Memetic Algorithm for University Exam Timetabling. In E. Burke and P. Roos, editors, *The Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference*, volume 1153 of *Lecture Notes in Computer Science*, pages 241–250. Springer-Verlag, Berlin, Germany, 1996.

[BPS99] S.C. Brailsford, C.N. Potss, and B.M. Smith. Constraint Satisfaction Problems: Algorithms and Applications. *European Journal of Operational Research*, 119:557–581, 1999.

[Bre76] J.A. Breslaw. A Linear Programming Solution to the Faculty Assignment Problem. *Socio-Economic Planning Science*, 10:227–230, 1976.

[Bre79] D. Brelaz. New Methods to Color the Vertices of a Graph. *Communications of the ACM*, 22(4):251–256, 1979.

[Bul98] B. Bullnheimer. An Examination Scheduling Model to Maximize Students' Study Time. In E. Burke and M. Carter, editors, *The Practice and Theory of Automated Timetabling: Selected Papers from the 2nd International Conference*, volume 1408 of *Lecture Notes in Computer Science*, pages 78–91. Springer-Verlag, Berlin, Germany, 1998.

[Car86] M.W. Carter. A Survey of Practical Applications of Examination Timetabling Algorithms. *Operations Research*, 34(2):193–202, 1986.

[CDM90] A. Colorni, M. Dorigo, and V. Maniezzo. Genetic Algorithms and Highly Constrained Problems: The Time-Table Case. In G. Goos and J. Hartmanis, editors, *Parallel Problem Solving from Nature*, Lecture Notes in Computer Science, pages 55–59. Springer-Verlag, Berlin, 1990.

[CDM92] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed Optimization by Ant Colonies. In F. J. Varela and P. Bourgine, editors, *The First European Conference on Artificial Life*, pages 134–142. MIT Press, Cambridge, MA, 1992.

[CDM+96] A. Colorni, M. Dorigo, F. Maffiolo, V. Maniezzo, G. Righini, and M. Trubian. Heuristics from Nature for Hard Combinatorial Optimization Problems. *International Transactions in Operational Research*, 3(1):1–21, 1996.

[CdW89] N. Chahal and D. de Werra. An Interactive System for Constructing Timetables on a PC. *European Journal of Operational Research*, 40:32–37, 1989.

[CG61] J. Csima and C.C. Gotlieb. Test on a Computer Method for Construction of School Timetables. *Communications of the ACM*, 7:160–163, 1961.

[CHS02] O. Cordon, F. Herrera, and T. Stützle. A Review of the Ant Colony Optimization Metaheuristic: Basis, Models and New Trends. *Mathware & Soft Computing*, 2-3:9:141–175, 2002.

[CK96] T.B. Cooper and J.H. Kingston. The Complexity of Timetable Construction Problems. In E. Burke and P. Roos, editors, *The Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference*, volume 1153 of *Lecture Notes in Computer Science*, pages 283–295. Springer-Verlag, Berlin, Germany, 1996.

[CKH02] P. Cowling, G. Kendall, and L. Han. An Investigation of a Hyperheuristic Genetic Algorithm to a Training Scheduling Problem. In *Proceedings of 2002 World Congress on Computational Intelligence (CEC 2002)*, pages 1182–1190. 2002.

[CKS00] P. Cowling, G. Kendall, and E. Soubeiga. A Hyperheuristic Approach to Scheduling a Sales Summit. In E. Burke and W. Erben, editors, *The Practice and Theory of Automated Timetabling: Selected Papers from the 3rd International Conference*, volume 2073 of *Lecture Notes in Computer Science*, pages 176–190. Springer-Verlag, Berlin, Germany, 2000.

[CKS01] P. Cowling, G. Kendall, and E. Soubeiga. Hyperheuristic: A Tool for Rapid Prototyping in Scheduling and Optimisation. In *Second European Conference on Evolutionary Computing for Combinatorial Optimisation (EvoCop 2002)*, pages 1–10. 2001.

[CL96] M.W. Carter and G. Laporte. Recent Developments in Practical Examination Timetabling. In E. Burke and P. Roos, editors, *The Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference*, volume 1153 of *Lecture Notes in Computer Science*, pages 3–21. Springer-Verlag, Berlin, Germany, 1996.

[CL98] M.W. Carter and G. Laporte. Recent Developments in Practical Course Timetabling. In E. Burke and M. Carter, editors, *The Practice and Theory of Automated Timetabling: Selected Papers from the 2nd International Conference*, volume 1408 of *Lecture Notes in Computer Science*, pages 3–19. Springer-Verlag, Berlin, Germany, 1998.

[Cos94] D. Costa. A Tabu Search for Computing an Operational Timetable. *European Journal of Operational Research*, 76:98–110, 1994.

[CP89] J. Carlier and E. Pinson. An Algorithm for Solving the Job-Shop Problem. *Management Science*, 35(2):164–176, 1989.

[CPU03] I. Cuevas, J. Panzera, and Y. Ugalde. Asignación de Salones y Horarios. *Proyecto de Grado - Taller V. Depto. de Investigación Operativa, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay*, 2003.

[CRF94] D. Corne, P. Roos, and H.L. Fang. Evolutionary Timetabling: Practice, Prospects and Work in Progress. In *UK Planning and Scheduling SIG Workshop*. 1994.

[CT89] M.W. Carter and C.A. Tovey. When is the Classroom Assignment Problem Hard? *Operations Research*, 40(1S):28–39, 1989.

[Dav91] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.

[DC99] M. Dorigo and G. Di Caro. The Ant Colony Optimization Meta-Heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw Hill, London, UK, 1999.

[DCG99] M. Dorigo, G. Di Caro, and L.M. Gambardella. Ant Algorithms for Discrete Optimization. *Artificial Life*, 5(2):137–172, 1999.

[DG97a] M. Dorigo and L.M. Gambardella. Ant Colonies for the Traveling Salesman Problem. *BioSystems*, 43:73–81, 1997.

[DG97b] M. Dorigo and L.M. Gambardella. Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem. *IEEE Transactions On Evolutionary Computation*, 1(1):53–66, 1997.

[DHS+88] M. Dincbas, P Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The Constraint Logic Programming Language CHIP. In *International Conference of 5th Generation Computer Systems (FGCS-88)*. ICOT, Tokyo, 1988.

[DJ91] T. Duncan and K. Johnson. Course Notes on Reasoning with Constraints. Technical report, Artificial Intelligence Applications Institute, University of Edinburh, 1991.

[DLU73] M.A.H Dempster, D.G. Lethridge, and A.M. Ulph. School Timetabling by Computer - A Technical History. Technical report, Oxford University, 1973.

[DM76] J. Dyer and J.M. Mulvey. The Implementation of an Integrated Optimization/Information System for Academic Department Planning. *Management Science*, 22:1332–1341, 1976.

[DMC91] M. Dorigo, V. Maniezzo, and A. Colorni. The Ant System: An Autocatalytic Optimizing Process. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1991.

[DMC92] M. Dorigo, V. Maniezzo, and A. Colorni. Positive Feedback as a Search Strategy. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992.

[DMC96] M. Dorigo, V. Maniezzo, and A. Colorni. The Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 26(1):29–41, 1996.

[DMV89] J.J. Dinkel, J. Mote, and M.A. Venkataramanan. An Efficient Decision Support System for Academic Course Scheduling. *Operations Research*, 37(6):853–864, 1989.

[Dor92] M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992. In Italian.

[DR90] V. Dhar and N. Ranganathan. Integer Programming vs. Expert Systems: An Experimental Comparison. *Communications of the ACM*, 33(3):323–336, 1990.

[DS01] M. Dorigo and T. Stützle. An Experimental Study of the Simple Ant Colony Optimization Algorithm. In C.E. D'Attellis et al., editor, *EC 2001: Proccedings of the 2001 WSES International Conference on Evolutionary Computation*. WSES-Press International, 2001.

[DS02] M. Dorigo and T. Stützle. The Ant Colony Optimization Metaheuristic: Algorithms, Applications and Advances. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research and Management Science*. Kluwer Academic Publishers, Boston, 2002.

[DS04] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, Massachusetts, 2004.

[Dun93] T. Duncan. A Review of Commercially Available Constraint Programming Tools. Technical Report AIAI-TR-149, Artificial Intelligence Applications Institute, University of Edinburgh, 1993.

[dW85] D. de Werra. An Introduction to Timetabling. *European Journal of Operational Research*, 19:151–162, 1985.

[DW87] R. Durbin and D.J. Willshaw. An Analogue Approach to the Travelling Salesman Problem using an Elastic Net Method. *Nature*, 38:591–598, 1987.

[dW96] D. de Werra. Some Combinatorial Models for Course Scheduling. In E. Burke and P. Roos, editors, *The Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference*, volume 1153 of *Lecture Notes in Computer Science*, pages 296–308. Springer-Verlag, Berlin, Germany, 1996.

[dW97] D. de Werra. The Combinatorics of Timetabling. *European Journal of Operational Research*, 96:513–513, 1997.

[ECF98] M.A.S. Elmohamed, P. Coddington, and G. Fox. A Comparison of Annealing Techniques for Academic Course Timetabling. volume 1408 of *Lecture Notes in Computer Science*, pages 92–112. Springer-Verlag, Berlin, Germany, 1998.

[EIS76] S. Even, A. Itai, and A. Shamir. On the Complexity of Timetabe and Multicommodity Flow Problems. *SIAM Journal of Computation*, 5:691–703, 1976.

[EK96] W. Erben and J. Keppler. A Genetic Algorithm Solving a Weekly Course-Timetabling Problem. In E. Burke and P. Roos, editors, *The Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference*, volume 1153 of *Lecture Notes in Computer Science*, pages 198–211. Springer-Verlag, Berlin, Germany, 1996.

[EL87] H.A. Eiselt and G. Laporte. Combinatorial Optimization Problems with Soft and Hard Requirements. *Journal of the Operational Research Society*, 38:785–795, 1987.

[Fer98] J.A. Ferland. Generalized Assignment-Type Problems: A Powerful Modelling Scheme. In E. Burke and M. Carter, editors, *The Practice and Theory of Automated Timetabling: Selected Papers from the 2nd International Conference*, volume 1408 of *Lecture Notes in Computer Science*, pages 53–77. Springer-Verlag, Berlin, Germany, 1998.

[FHL96] J.A. Ferland, A. Hertz, and A. Lavoie. An Object-Oriented Methodology for Solving Assignment-Type Problems with Neighbourhood Search Techniques. *Operation Researchs*, 44:347–359, 1996.

[FL92] J.A. Ferland and A. Lavoie. Exchange Procedures for Timetabling Problems. *Discrete Applied Mathematics*, 35:237–253, 1992.

[FM94] E.C. Freuder and A.K. Mackworth. Constraint-Based Reasoning. *MIT/Elsevier, Cambridge*, 1994.

[FR85] J.A. Ferland and S. Roy. Timetabling Problem for University as Assignment of Activity to Resources. *Computers and Operational Research*, 12(2):207–218, 1985.

[FRL86] J.A. Ferland, S. Roy, and T.G. Loc. The Timetabling Problem. *Models on Microcomputers*, pages 97–103, 1986. North-Holland.

[FS83] J.G. Fisher and D.R. Shier. A Heuristic Procedure for Large-Scale Examination Scheduling Problems. Technical Report 417, Department of Mathematical Sciences, Clemson University, 1983.

[FW91] B. Fritzke and P. Wilke. Flexmap - A Neural Network for the Travelling Salesman Problem with Linear Time and Space Complexity. Technical report, Universität Erlangen-Nürnberg, 1991.

[GAD89] S. Goss, S. Aron, and J.L. Deneubourg. Self-Organized Shortcuts in the Argentine Ant. *Naturwissenschaften*, 76:579–581, 1989.

[GD96] L.M. Gambardella and M. Dorigo. Solving Symmetric and Asymmetric TSPs by Ant Colonies. In E. Burke and M. Carter, editors, *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation (ICEC'96)*, pages 622–627. IEEE Press, Piscataway, NJ, 1996.

[Gen02] M. Gendreau. Recent Advances in Tabu Search. In F.Glover and G.A. Kochenberger, editors, *Essays and Surveys in Metaheuristics*, pages 369–378. Kluwer Academic Publisher, 2002.

[Gen03] M. Gendreau. An Introduction to Tabu Search. In F. Glover and G.A. Kochenberger, editors, *Handbooks of Metaheuristics*, pages 37–54. Kluwer Academic Publisher, 2003.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability - A Guide to $\mathcal{NP}$-Completeness*. W.H. Freeman and Company, San Francisco, 1979.

[GJBP96] C. Gueret, N. Jussien, P. Boizumault, and C. Prins. Building University Timetables using Constraint Logic Programming. In E. Burke and P. Roos, editors, *The Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference*, volume 1153 of *Lecture Notes in Computer Science*, pages 130–145. Springer-Verlag, Berlin, Germany, 1996.

[GKM90] E. Gudes, T. Kuflik, and A. Meisels. On Resource Allocation by an Expert System. *Engineering Applications of Artificial Intelligence*, 3:101–109, 1990.

[GL93] F. Glover and M. Laguna. Tabu Search, Modern Heuristic Techniques for Combinatorial Problems. In C.R. Reeves, editor, *Blackwell Scientific Publications, Oxford*, pages 70–150. Oxford, 1993.

[GL97] F. Glover and M. Laguna. Tabu Search. 1997.

[GL02] F. Glover and M. Laguna. Tabu Sarch. In P.M. Pardalos and M.G.C. Resende, editors, *Handbook of Applied Optimization*, pages 194–208. Oxford University Press, New York, 2002.

[Glo77] F. Glover. Heuristics for Integer Programming using Surrogate Constraints. *Decision Sciences*, 8:156–166, 1977.

[Glo89] F. Glover. Tabu Search: Part I. *ORSA Journal on Computing*, 1:190–206, 1989.

[Glo90] F. Glover. Tabu Search: Part II. *ORSA Journal on Computing*, 3:223–254, 1990.

[Glo97] F. Glover. Tabu Search and Adaptative Memory Prgramming - Advances, Applications and Challenges. In R.S. Barr, R.V. Helgason, and J.L. Kennington, editors, *Advances in Metaheuristics, Optimization and Stochastic Modeling Technologies*, pages 1–75. Kluwer Academic Publisher, 1997.

[GLT93] F. Glover, M. Laguna, and E.D. Taillard. Tabu Sarch. *Annals of Operations*, 41, 1993.

[Gol89] D.E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley, 1989.

[Gol00] H.J. Goltz. On Methods of Constraint-Based Timetabling. In *Proceedings of PACLP'2000*, pages 167–177. 2000.

[Got63] C.C. Gotlieb. The Construction of Class-Teacher Timetables. *IFIP Congress*, 62:73–77, 1963.

[GPS89] L. Gislen, C. Peterson, and B. Soderberg. Teachers and Classes with Neural Networks. *International Journal of Neural Systems*, 1(2):167–176, 1989.

[GPS92] L. Gislen, C. Peterson, and B. Soderberg. Complex Scheduling with Potts Neural Networks. *Neural Computation*, 1:805–831, 1992.

[HdW87] A. Hertz and D. de Werra. Using Tabu Search Techniques for Graph Colouring. *Computing*, 39:345–351, 1987.

[Her91] A. Hertz. Tabu Search for Large Scale Timetabling Problems. *European Journal of Operational Research*, 54:39–47, 1991.

[Hil81] A.J. Hilton. School Timetables. *Annals of Discrete Mathematics*, 11:177–188, 1981.

[HK62] M. Held and R.M. Karp. A Dynamic Programming Approach to Sequencing Problem. *SIAM*, 1962.

[Hol75] J.H. Holland. *Adaption in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan, 1975.

[HT85] J.J. Hopfield and D. Tank. Neural Computation for Decisions in Optimization Problems. *Biological Cybernetics*, 5:141–152, 1985.

[HW96] M. Henz and J. Wurtz. Using Oz for College Timetabling. In E. Burke and P. Roos, editors, *The Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference*, volume 1153 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 1996.

[Joh93] D. Johnson. A Database Approach to Course Timetabling. *Journal of the Operational Research Society*, 44:425–433, 1993.

[Jun86] W. Junginger. Timetabling in Germany - A Survey. *Interfaces*, 16:66–74, 1986.

[KG96] M. Kambi and D. Gilber. Timetabling in Constraint Logic Programming. In *Proceedings of the INAP-96: Symposium and Exhibition on Industrial Applications of Prolog*, pages 79–88, 1996.

[KGV83] S. Kirkpatrick, C.D. Gelatt, and P.M. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.

[KKK97] L.F. Kwok, S.C. Kong, and Y.Y. Kam. Timetabling in Hong Kong Secondary Schools. *Computer & Education*, 28:173–183, 1997.

[Kle83] D. Klein. *The Application of Computerized Solution Techniques to the Problem of Timetabling in High Schools and Universities.* PhD thesis, Concordia University, 1983.

[Koz92] J.R. Koza. Genetic Programming. On the Programming of Computers by Means of Natural Selection. *The MIT Press*, 819, 1992.

[KW92] L. Kang and G.M. White. A Logic Approach to the Resolution of Constraints in Timetabling. *European Journal of Operational Research*, 61:306–317, 1992.

[Laj96] G. Lajos. Complete University Modular Timetabling using Constraint Logic Programming. In E. Burke and P. Roos, editors, *The Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference*, volume 1153 of *Lecture Notes in Computer Science*, pages 148–161. Springer-Verlag, Berlin, Germany, 1996.

[LD86] G. Laporte and S. Desroches. The Problem of Assigning Students to Course Sections in a Large Engineering School. *Computers and Operational Research*, 13:387–394, 1986.

[Len86] M.J. Lennon. Examination Timetabling at the University of Auckland. *New Zealand Operational Research*, 14:176–178, 1986.

[Lin92] S. Ling. Integrating Genetic Algortihms with a Prolog Assignment Problem as an Hybrid Solution for a Polytechnic Timetable Problem. In R. Manner and B. Manderick, editors, *Parallel Problem Solving from Nature II*, pages 321–329. Elsevier Science Publisher, 1992.

[LK91] A.M. Law and W.D. Kelton. *Simulation Modeling & Analysis.* McGraw-Hill, New York, 1991.

[MC91] D.F.X. Mathaisel and C.L. Comm. Course and Classroom Scheduling - An Interactive Computer-Graphics Approach. *Journal of Systems and Software*, 15(2):149–157, 1991.

[MCR98] F. Melicio, J.P. Caldeira, and A.C. Rosa. Timetabling Implementation Aspects by Simulated Annealing. In J. Gu, editor, *IEEE - Systems Science and Systems Engineering*, pages 553–557. 1998.

[MEsG93] A. Meisels, J. Ell-sana, and E. Gudes. Comments on CSP Algorithms Applied to Timetabling. Technical report, Department of Mathematics and Computer Science, Ben-Gurion University, Israel, 1993.

[MF75] G. McMahon and M. Florian. On Scheduling with Ready Times and Due Dates to Minimize Maximum Lateness. *Operations Research*, 23(3):475–482, 1975.

[MGK91] A. Meisels, E. Gudes, and T. Kuflik. Limited Resource Time-Tabling by a Generalized Expert System. *Knowledge-Based Systems*, 4:215–224, 1991.

[MGS96] A. Meisels, E. Gudes, and G. Solotorevsky. Employee Timetabling, Constraint Networks and Knowledge-Based Rules: a Mixed Approach. In E. Burke and P. Roos, editors, *The Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference*, volume 1153 of *Lecture Notes in Computer Science*, pages 93–105. Springer-Verlag, Berlin, Germany, 1996.

[MN92] P. Moscato and M.Ñorman. A Memetic Approach for the Travelling Salesman Problem – Implementation of a Computational Ecology for Combinatorial Optimisation on Message Passing Systems. In *Proceedings of the International Conference on Parallel Computing and Transputer Applications*, pages 177–186. IOS Press, Amsterdam, 1992.

[Mon88] A. Monfroglio. Time-Tabling Through a Deductive Database: A Case Study. *Data and Knowledge Engineering*, 3:1–27, 1988.

[MPL90] S. Minton, A. Phillips, and P. Laird. Solving Largescale CSP and Scheduling Problems using a Heuristic Repair Method. *In Proceedings of the 8th AAAI Conference*, pages 17–24, 1990.

[MRTT53] N. Metrópolis, M. Rosenbluth, A. Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[MW84] R.H. McClure and C.E. Wells. A Mathematical Programming Model for Faculty Course Assignment. *Decision Science*, 15:409–420, 1984.

[NI98] K.Ñonobe and T. Ibaraki. A Tabu Search Approach to the Constraint Satisfaction Problem as a General Problem Solver. *European Journal of Operational Research*, 106(2-3):599–623, 1998.

[OdW83] R. Ostermann and D. de Werra. Some Experiments with a Timetabling Problem. *OR Spektrum*, 3:199–204, 1983.

[PCL95] B. Paechter, A. Cumming, and H. Luchian. The use of Local Search Suggestion Lists for Improving the Solution of Timetabling Problems with Evolutionary Algorithms. In G. Goos, J. Hartmanis, and J. Leeuwen, editors, *Evolutionary Computation, AISB Workshop*, volume 993 of *Lecture Notes in Computer Science*, pages 86–93. Springer-Verlag, Berlin, 1995.

[PCNL96] B. Paechter, A. Cumming, M.G. Norman, and H. Luchian. Extensions to a Memetic Timetabling System. In E. Burke and P. Roos, editors, *The Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference*, volume 1153 of *Lecture Notes in Computer Science*, pages 251–266. Springer-Verlag, Berlin, Germany, 1996.

[PCSD89] C. Petrie, R. Causey, D. Steiner, and V. Dhar. A Planning Problem: Revisable Academic Course Scheduling. Technical Report ACT-AI-020, Microelectronics and Computer Technology Corporation, 1989.

[PDG87] J.M. Pasteels, J.L. Deneubourg, and S. Goss. Self-Organization Mechanisms in Ant Societies (i): Trail Recruitment to Newly Discovered Food Sources. *Experientia Supplementum*, 54:155–175, 1987.

[Pea84] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Reading, MA, 1984.

[PRC98] B. Paechter, B. Rankin, and A. Cumming. Improving a Lecture Timetabling System for University-wide use. In E. Burke and M. Carter, editors, *The Practice and Theory of Automated Timetabling: Selected Papers from the 2nd International Conference*, volume 1408 of *Lecture Notes in Computer Science*, pages 156–168. Springer-Verlag, Berlin, Germany, 1998.

[Ran96] R.C. Rankin. Automatic Timetabling in Practice. In E. Burke and P. Roos, editors, *The Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference*, volume 1153 of *Lecture Notes in Computer Science*, pages 266–282. Springer-Verlag, Berlin, Germany, 1996.

[RC95] P. Ross and D. Corne. Comparing Genetic Algorithms, Simulated Annealing and Stochastic Hill-Climbing on Ttimetable Problems. In G. Goos, J. Hartmanis, and J. Leeuwen, editors, *Evolutionary Computation, AISB Workshop*, volume 993 of *Lecture Notes in Computer Science*, pages 94–102. Springer-Verlag, Berlin, 1995.

[RC99] F. Rey and M. Carlevaro. Problema de Asignación de Horarios y Salones a Cursos. *Proyecto de Grado - Taller V. Depto. de Investigación Operativa, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay*, 1999.

[RDSC+02] O. Rossi-Doria, M. Sampels, M. Chiarandini, J. Knowles, M. Manfrin, M. Mastrolilli, L. Paquete, and B. Paechter. A Comparison of the Performance of Diferent Metaheuristics on the Timetabling Problem. In E. Burke and P. De Causmaecker, editors, *The Practice and Theory of Automated Timetabling: Selected Papers from the 4th International Conference*, Lecture Notes in Computer Science, pages 124–127. Springer-Verlag, Berlin, Germany, 2002.

[RJ93] P. Ross and J.Hallam. Lecture Notes on Conectionist Computing. Technical report, Department of Artificial Inteligence, University of Edinburgh, 1993.

[RS94] N. Radcliffe and P. Surry. Formal Memetic Algorithms. volume 865 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, Berlin, 1994.

[RS96] V. Ram and C. Scogings. Automated Timetable Generation Using Multiple Context Reasoning with Truth Maintenance. In E. Burke and P. Roos, editors, *The Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference*, volume 1153 of *Lecture Notes in Computer Science*, pages 106–111. Springer-Verlag, Berlin, Germany, 1996.

[Sch95] A. Schaerf. A Survey of Automated Timetabling. Technical Report CS-R9567, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, The Netherlands, 1995.

[Sch96] A. Schaerf. Tabu Search for Large High-School Timetabling Problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 29(4):368–377, 1996.

[SGM94] G. Solotorevsky, E. Gudes, and A. Meisels. RAPS: A Rrule-Based Language Specifying Resource Allocation and Time-Tabling Problems. *IEEE Transactions on Knowledge and Data Engineering*, 6(5):681–697, 1994.

[SH96] T. Stützle and H.H. Hoos. Improving the Ant System: A Detailed Report on the $\mathcal{MAX} - \mathcal{MIN}$ Ant System. Technical Report AIDA-96-12, FG Intellektik, FB Informatik, TU Darmstadt, Germany, 1996.

[SH97] T. Stützle and H.H. Hoos. $\mathcal{MAX} - \mathcal{MIN}$ Ant System and Local Search for the Traveling Salesman Problem. In T. Bäck, Z. Michalewicz, and

X. Yao, editors, *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (ICEC'97)*, pages 309–314. IEEE Press, Piscataway, NJ, 1997.

[SH00] T. Stützle and H.H. Hoos. $\mathcal{MAX} - \mathcal{MIN}$ Ant System. *Future Generation Computer Systems*, 16(8):889–914, 2000.

[SK90] J. Skorin-Kapov. Tabu Search Applied to the Quadratic Assignment Problem. *ORSA Journal on Computing*, 2:34–45, 1990.

[SKS02] K. Socha, J. Knowles, and M. Sampels. A $\mathcal{MAX} - \mathcal{MIN}$ Ant System for the University Course Timetabling Problem. In Marco Dorigo, Giani Di Caro, and Michael Sampels, editors, *Proceedings of ANTS 2002 – Third International Workshop on Ant Algorithms*, volume 2463 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, Berlin, Germany, September 2002.

[SS77] W. Shin and J.A. Sullivan. Dynamic Course Scheduling for College Faculty Via Zero-One Programming. *Decision Science*, 8:711–721, 1977.

[SS79] G. Schmidt and T. Strohein. Timetable Construction - An Annotated Bibliography. *The Computer Journal*, 23(4):307–316, 1979.

[Stü98] T. Stützle. *Local Search Algorithms for Combinatorial Problems: Analysis, Improvements and New Applications*. PhD thesis, Intellectics Group, Department of Computer Science, Darmstadt University of Technology, Germany, 1998.

[Stü99] T. Stützle. *Local Search Agorithms for Combinatorial Problems: Analysis, Improvements, and New Applications*. Infix, Sankt Augustin, Germany, 1999.

[SVK97] P. Stamatopoulus, E. Viglas, and S. Karaboyas. Nearly Optimum Timetable Construction through CLP and Intelligent Search. *International Journal on Artificial Intelligence Tools*, 7(4):415–442, 1997.

[Tai90] E. Taillard. Robust Tabu Search for the Quadratic Assignment Problem. Technical Report ORPW 90/10, Swiss Federal Institute of Technology of Lausanne, 1990.

[TD98] J.M. Thompson and K.A. Dowsland. A Robust Simulated Annealing based Examination Timetabling System. *Computers and Operations Research*, 25:637–648, 1998.

[Tri80] A. Tripathy. A Lagrangean Relaxation Approach to Course Timetabling. *Journal of the Operational Research Society*, 31:599–603, 1980.

[Tri84] A. Tripathy. School Timetabling - A Case in Large Binary Integer Linear Programming. *Management Science*, 30(12):1473–1489, 1984.

[Tri92] A. Tripathy. Computerized Decision Aid for Timetabling - A Case Analysis. *Discrete Applied Mathematics*, 35(3):313–323, 1992.

[TRV99] H. Terashima, P. Ross, and M. Valenzuela. Evolution of Constraint Satisfaction Strategies in Examination Timetabling. In Morgan Kaufmann, editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, pages 635–642. 1999.

[Tsa93] E.P.K. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, London and San Diego, 1993.

[TW99] E. Tsang and R. Williams. A Computer Aided Constraint Programming System. *The First International Conference on the Practical Application of Constraint Technologies and Logic Programming*, pages 81–93, 1999.

[Vin84] P. Vincke. Timetabling in Belgium: A Survey. In *TIMS XXVI*. Copenhagen, 1984.

[vLA87] P.J.M. van Laarhoven and E.H.L. Aarts. *Simulated Annealing: Theory and Applications*. D. Reidel Publishing Company, Kluwer Academic Publishers Group, 1987.

[WC79] G.M. White and P.W. Chan. Towards the Construction of Optimal Examination Timetables. *INFOR*, 17:219–229, 1979.

[Whi00] G.M. White. Constrained Satisfaction, Not so Constrained Satisfaction and the Timetabling Problem. In E. Burke and W. Erben, editors, *The Practice and Theory of Automated Timetabling: Selected Papers from the 3rd International Conference*, volume 2073 of *Lecture Notes in Computer Science*, pages 32–47. Springer-Verlag, Berlin, Germany, 2000.

[WN90] K.H. Wong and W.Y. Ng. An Interactive Timetabling Support System. In *Int. Conf. in System Management*, pages 307–313. Honk Kong, 1990.

[Wre81] In A. Wren, editor, *Computer Scheduling of Public Transport*. North-Holland, 1981.

[Wre96] A. Wren. Scheduling, Timetabling and Rostering – A Special Relationship? In E. Burke and P. Roos, editors, *The Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference*, volume 1153 of *Lecture Notes in Computer Science*, pages 46–75. Springer-Verlag, Berlin, Germany, 1996.

[WZ97] G.M. White and J. Zhang. Generating Complete University Timetables by Combining Tabu Search with Constraint Logic. In E. Burke and M. Carter, editors, *The Practice and Theory of Automated Timetabling: Selected Papers from the 2nd International Conference*, volume 1408 of *Lecture Notes in Computer Science*, pages 187–200. Springer-Verlag, Berlin, Germany, 1997.

[YK92] H. Yokoi and Y. Kakazu. An Approach to the Travelling Salesman Problem by a Bionic Model. *Heuristic: the journal of knowledge engineering*, 5:13–27, 1992.

[YMNW94] M. Yoshikawa, K. Maneko, Y.Ñomura, and M. Watanabe. A Constraint-Based Approach to High-School Timetabling Problems: A Case Study. In *Proceedings of the twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 1111–1116, 1994.