Proyecto de Grado - Taller V

Problema de Asignación de Horarios y Salones a Cursos

Fernando Rey - Marcos Carlevaro

Tutor: Dr. Ing. Héctor Cancela
Depto. de Investigación Operativa

Instituto de Computación
Facultad de Ingeniería
Universidad de la República

Indice

1 Introducción 5

2 Relevamiento y Planteo del Problema a Resolver 7

2.1 Relevamiento 7

2.2 Descripción en Lenguaje Natural del Problema 7

2.3 Entrevistas con Docentes de la Facultad 10

2.4 Clasificación Problema y Estudios realizados sobre problemas similares 10

3 Formalización del Problema 12

3.1 Planteo General de un Problema de Optimización 12

3.2 Descripción del Problema como un Problema de Optimización 13

3.2.1 Restricciones 14
3.2.2 Preferencias 15
3.2.3 Función Objetivo 16

3.3 Formalización 17

3.3.1 Definición de Entidades 17
3.3.2 Restricciones 18
3.3.3 Preferencias 20
3.3.4 Función Objetivo 26

3.4 Complejidad de Nuestro Problema 26

3.4.1 Problemas NP- Completos 26
3.4.2 Problema de Asignación Cuadrático (QAP - Quadratic Assignment Problem) 27
3.4.3 Analogía con el QAP 29
3.4.4 Naturaleza Cuadrática de la función objetivo del Problema de Asignación 29

4 Estudio de Técnicas y su Posible Aplicación al Problema 31

4.1 Técnicas Exactas 31

4.2 Heurísticas 32

4.2.1 Ant Systems - Colonias de Agentes Cooperativos 32
4.2.2 Algoritmos genéticos 38
4.2.3 Simulated Annealing 39
4.2.4 Sampling and Clustering 40
4.2.5 Tabu Search 41
4.2.6 Neural Nets 43

5 Diseño utilizando Ant Systems 44

5.1 Resolución del QAP utilizando la heurística Ant Systems 44

5.2 Resolución del problema de asignación utilizando Ant Systems...47

5.2.1 Analogía con el QAP 47

5.2.2 Representación del Problema de Asignación de Salones y Horarios a Cursos 47

6 Implementación 55

6.1 Lenguaje de programación a utilizar 55

6.2 Estructuras de datos.....55

6.2.1 Estructura de la facultad.....56

6.2.2 La hormiga 58

6.2.3 Rastros.....59

6.2.4 Asignación.....60

6.3 Formato de los datos de entrada 60

6.3.1 Archivo de clases 61

6.3.2 Archivo de salones 61

6.3.3 Archivo de estructura de la facultad 62

6.3.4 Archivo de parámetros 64

6.4 Formato de los datos de salida.....65

6.5 Principales elementos del algoritmo 66

6.5.1 Pseudocódigo del algoritmo principal 66

6.5.2 Pseudocódigo de la función “mover” que ejecuta la hormiga 67

6.5.3 Pseudocódigo de la función “elegir clase” 69

6.6 Decisiones de implementación 70

6.6.1 Hormigas.....70

6.6.2 Restricciones y preferencias 70

6.6.3 Elección de una clase 71

6.6.4 Costo de la asignación 72

6.6.5 Rastros y evaporación.....73

7 Pruebas(I) – Juego de datos ficticio 74

7.1 Juego de datos de prueba ficticios 74

7.2 Pruebas y resultados 74

7.3 Ajuste de parámetros.....85

7.4 Conclusiones extraídas del estudio de las pruebas con el juego de datos ficticio 87

8 Variantes y mejoras al algoritmo básico 88

8.1 Nuevos parámetros 88

8.2 Mejor hormiga.....90

8.3 Asignación en bloque 90

9 Pruebas(II) – Juego de datos reales..... 92

9.1 Juego de datos de prueba reales 92
9.1.1 Maxigrupos 92
9.1.2 Utilización de los resultados de la asignación 93
9.1.3 Resumen del juego de datos reales 94
9.1.4 Variantes en las características de los juegos de datos 95

9.2 Pruebas y resultados 96
9.2.1 Descripción del subconjunto de datos utilizado 96
9.2.2 Ejemplos de pruebas realizadas 97

9.3 Ajuste de parámetros y mejoras al algoritmo 105
9.3.1 Ajuste de parámetros 105
9.3.2 Mejoras al algoritmo 105

9.4 Pruebas utilizando las mejoras efectuadas 113

9.5 Conclusiones extraídas del estudio de las pruebas con los juegos de datos reales 118

10 Conclusiones finales y trabajo futuro 120

11 Cronograma 121

12 Bibliografía 122

Apéndice A – Juego de datos reales 124

Apéndice C – Fuentes 154

Fuentes del algoritmo Ant System clásico 154

Cambios en los fuentes del algoritmo para implementar la mejora "mejor hormiga" 263

Cambios en los fuentes del algoritmo para implementar mejoras a las funciones de visibilidad y rastros 295

Problema de Asignación de Salones y Horarios a Cursos

Resumen

De año en año la Facultad de Ingeniería, así como otras facultades, liceos o institutos, debe enfrentarse con el problema de asignar los salones y horarios para cada curso que se dictará en los meses siguientes. Este problema se resuelve actualmente en forma manual y lleva varias horas de trabajo. Sería útil entonces encontrar una manera de automatizar esta tarea para facilitarla y ahorrar así tiempo y esfuerzo. El problema a tratar es un problema de optimización combinatoria complejo, y se consideran varias alternativas para su resolución, en particular las metodologías heurísticas desarrolladas a partir de la observación de fenómenos de la naturaleza. Dentro de esta familia de algoritmos, se estudia la aplicación al problema de una heurística basada en el comportamiento de colonias de agentes cooperativos, conocida como 'Ant Systems'. Este método no ha sido aplicado aún a problemas de asignación en instituciones de enseñanza.

1 Introducción

En el área de la Investigación Operativa y dentro de ésta considerando los problemas de optimización combinatoria, han surgido hace algunos años métodos de resolución heurísticos. Esto fue motivado por la complejidad de algunos de estos problemas, NP-completos, los cuales no resulta práctico resolverlos por técnicas exactas, ya que llevarían un tiempo de procesamiento demasiado extenso.

Con la utilización de métodos aproximados, se han logrado buenos resultados en tiempos razonables. Es así que ha surgido entonces una nueva 'familia' de heurísticas, cuya idea básica es imitar fenómenos o comportamientos existentes en la naturaleza, las cuales logran buenos resultados en base a la exploración del espacio y las aproximaciones sucesivas.

La tarea de asignar los salones y horarios disponibles en el espacio de una semana para los cursos a dictarse en un período dado en la Facultad es compleja por varias razones. Dada la 'escasez' de salones, horarios y profesores para abarcar todos los cursos que deben ser dictados, es difícil darles una buena ubicación a todos. Por otra parte, existe gran cantidad de requerimientos que la asignación debe cumplir para ser satisfactoria.

Actualmente, la asignación es realizada manualmente por docentes o funcionarios de la Facultad, tarea que puede llevar varios días. En el Instituto de Matemática y Estadística de la Facultad se han llevado a cabo algunos estudios para resolver el

problema por algún método algorítmico, pero no se ha llegado a concretar una solución.

El objetivo del presente trabajo es implementar una solución al problema de asignación de salones y horarios a cursos en la Facultad de Ingeniería utilizando la heurística Ant Systems. Las etapas principales de las que consta el proyecto son: Primero realizar un relevamiento del problema, estudiando las condiciones que debe cumplir una buena asignación. Luego se planteará el mismo como un problema de optimización combinatoria. Finalmente se diseñará e implementará una solución. Por otra parte, se realizará un estudio de otras técnicas que posibiliten resolver el problema en un tiempo razonable, en especial técnicas heurísticas.

La heurística 'Ant Systems' se basa en el comportamiento de 'colonias de agentes cooperativos'. En palabras simples, ésta imita el comportamiento de las hormigas para encontrar buenos caminos entre dos puntos. Esta técnica ha sido aplicada con éxito a problemas de optimización tipo, pero no ha sido aplicada aún a problemas como el considerado.

Este informe está organizado de la siguiente manera: La sección 2. describe en forma detallada el problema a resolver. La sección 3 permite ver al mismo como un problema de optimización. La sección 4 estudia distintas alternativas para implementar una solución, y en la sección 5 se estudia en particular la aplicación de 'Ant Systems'. La implementación utilizando Ant Systems se describe en la sección 6, y en la 7 se describe un primer conjunto de pruebas realizadas y resultados obtenidos. Estos resultados se fueron mejorando a partir de las modificaciones al algoritmo descritas en la sección 8. Un segundo conjunto de pruebas se analiza en la sección 9. La sección 10 contiene las conclusiones de todo el trabajo realizado y las perspectivas de trabajo futuro. La sección 11 contiene un cronograma con el tiempo invertido en cada tarea . La bibliografía utilizada se encuentra en la sección 12.

2 Relevamiento y Planteo del Problema a Resolver

2.1 Relevamiento

Las características y datos particulares del problema de asignación en la facultad se obtuvieron mediante el aporte de profesores de la facultad que habían tenido contacto con el tema, y por las observaciones de los propios autores de este trabajo, estudiantes de la misma.

2.2 Descripción en Lenguaje Natural del Problema

El problema a resolver consiste en asignar salones de clase a los cursos a dictarse en determinado período en una facultad, teniendo en cuenta las características de los salones y los requerimientos de los cursos.

Se considerará la asignación de los cursos a dictarse en el período de un semestre. Parece que sería adecuado asignar clases por semestre y no por año ya que es más estática la estructura de cursos en un semestre y por lo tanto más manejable. Es raro que existan cursos que duren menos de un semestre, y los cursos anuales pueden dividirse en dos semestrales.

Se asignará la extensión horaria correspondiente a una semana de clases debido a que la casi totalidad de los cursos tiene frecuencia semanal.

No está dentro del alcance de este problema la asignación de docentes a cursos. Consideramos que ese problema está resuelto previamente.

Las principales entidades a considerar en el problema son:

- Salones
- Cursos
- Horarios

Para cada entidad consideramos los atributos relevantes:

Salones:

- Capacidad locativa. Nos interesará asignar una clase a un salón que pueda alojar la totalidad de los estudiantes que concurran a ella.
- Otras características: bajo este atributo se considera una lista de características del salón como por ejemplo si tiene cortinas, nro. de enchufes, tipo de pizarrón, aire acondicionado, etc. Existen cursos para los cuales es necesario que el salón que se le asigne cumpla algunas de estas características.

No consideramos en principio el hecho de que un salón disponga de computadoras, ya que en la facultad este atributo solo se encuentra en algunos salones en particular, y la asignación de estos salones podría considerarse como un subproblema aparte,

dadas sus características muy diferentes a las de un salón común, y dado que tienen una asignación mucho más libre y variable en el transcurso del año.

Cursos:

Un curso o materia consta de una o más **clases** semanales de cierta duración. En principio podríamos pensar en asignar todas las clases de un curso sin distinguirlas unas de otras, pero existen muchos casos en que estas clases tienen características o necesidades diferentes unas de otras. Por lo tanto, se llegó a la conclusión de que es necesario considerar las CLASES que se dictarán semanalmente de un curso, y no solamente el curso en sí. Por ejemplo, en un curso de 2 clases semanales, podría ocurrir que tuvieran distinta preferencia de horarios, distinta duración, o que una de las clases requiriera el uso de retroproyector.

Ejemplo: El curso de Teoría de la Programación 1 consta de 2 bloques temáticos independientes que son dictados por 2 profesores diferentes pero el dictado de estos 2 grandes temas no se hace en forma secuencial, es decir no termina uno de los temas antes de comenzar el otro sino que los martes y jueves uno de los profesores dicta el tema Teoría de Autómatas en un horario de las 7 de la tarde y por otro lado los viernes el otro profesor el tema Computabilidad y complejidad en el horario de la mañana.

Continuando con el estudio del problema, vemos que la facultad está organizada en una estructura jerárquica:

- Facultad
    - Carreras
        - Años
            - Cursos
                - Grupos
                    - Clases

La facultad consta de distintas carreras (por ejemplo Civil, Mecánica, Computación), que muchas veces comparten algunos cursos. Cada carrera tiene una duración de varios años. En cada año se dictan un conjunto de cursos. Estos cursos, ya sean teóricos o prácticos, pueden dividirse en grupos, con el fin de organizar a los estudiantes y hacer clases menos numerosas. Cada grupo consta de un número de clases.

A continuación caracterizamos las entidades que tienen más importancia en nuestro problema:

Cursos:

- nivel de importancia: algunos cursos tendrán prioridad sobre otros debido a que son básicos para una o más carreras, y son cursados por un número importante de estudiantes (por ej. Análisis II, Algebra, Mecánica).
- carga horaria y cantidad de clases por semana.
- cantidad estimada de estudiantes.

Clases:

- preferencia de horarios: cada clase dispone de un conjunto de rangos horarios dentro de los cuales puede ser asignada, debido ya sea al año que corresponde, a la disponibilidad del docente o a las necesidades de los estudiantes.
- duración.
- cantidad estimada de estudiantes.
- otras características (análogo a las “otras características” del salón)
- carreras a las que pertenece
- años a los que pertenece
- cursos a los que pertenece
- grupos a los que pertenece
- si es teórico o es de práctico

Horarios:

Durante la semana los horarios en que se imparten clases van desde las 8:00 a las 22:00, salvo raras excepciones, de lunes a viernes, a veces sábados.
Existen horarios con más demanda, como por ej. de 8:00 a 10:00 o de 16:00 a 18:00, y algunos con poca demanda, por ej. de 12:00 a13:00.
No existen clases cuya duración sea menor a media hora, en promedio la misma es de una hora y media, llegando en algunos casos hasta tres horas.

Condiciones que debe cumplir la asignación:

La asignación de los salones a cursos debe cumplir diversas condiciones para ser satisfactoria. Algunas de ellas son más rígidas ya que si no se cumplen la asignación no será valida. Otras solo expresan preferencias referentes a cursos, horarios, salones o combinaciones de ellas.

- Todos los cursos deben tener un salón asignado, o al menos los cursos más importantes deben estar asignados, o al menos algunas clases de un curso deben estar asignadas.
- Dentro de un grupo no pueden existir clases que se superpongan.
- En ciertas carreras como ingeniería mecánica, existen cursos pertenecientes a distintos años que no es deseable que se superpongan. Esto se debe a que existe un numero elevado de estudiantes que cursan al mismo tiempo materias de distintos años.
- Espaciamiento de las clases de un mismo grupo, en distintos días de la semana. No es conveniente que las clases a dictar en la semana se den en días consecutivos. Por ej., si un curso tiene tres clases de teórico, es bueno que se dicten lunes, miércoles y viernes.
- Las clases de un mismo grupo deben asignarse a la misma hora en distintos días.
- Clases asignadas a salones de capacidad adecuada.
- Clases asignadas a salones con “otras características” compatibles. Por ejemplo, base de datos debe darse con transparencias y no puede impartirse en un salón sin enchufe.
- Que los prácticos se dicten en las adyacencias del teórico correspondiente.
- Dado un curso que no todos grupos se encuentren comprendidos en una reducida franja horaria. Por ejemplo de análisis II se precisa que hayan 2 grupos en horas de la tarde y un grupo en horas de la noche. Para álgebra como para análisis se espera que hayan cursos de teórico de mañana, de tarde, de noche.
- Que las clases de un mismo grupo no se dicten en turnos diferentes.

- Considerar el hecho de materias donde se cuenta con un solo docente posible de teórico debido a la escasez de docentes del nivel necesario que estén dispuestos a impartirla, aquí las restricciones dependen fuertemente de las disponibilidades del docente y por consiguiente son muchas veces insoslayables.
- Existen recursos que no necesariamente están relacionados con un salón en particular. Por ejemplo, la facultad cuenta con un número finito de retroproyectores; puede darse el problema que en una franja horaria se asignen salones a cursos de modo que no alcancen los retroproyectores.

El objetivo al realizar la asignación es que esta cumpla las condiciones más rígidas y satisfaga de la mejor manera posible las más 'blandas'.

Una utilidad extra del programa sería su uso como una herramienta de ayuda en la toma decisiones. Por ejemplo podría ser ejecutarlo para ambos semestres considerando distintas opciones de acomodar ciertos cursos en los semestres y comparar las asignaciones resultantes eligiendo la más conveniente. Esto puede ayudar a decidir en qué semestre se dictará un curso dependiendo de la mayor o menor disponibilidad de salones en las respectivas asignaciones.

2.3 Entrevistas con Docentes de la Facultad

El Instituto de Matemáticas (IMERL) de la Facultad de Ingeniería estuvo trabajando en la asignación de salones a cursos, principalmente la formación de grupos de Análisis I, II y Algebra. En relación con este trabajo, la Prof. Graciela Ferreira del INCO aportó algunos datos de la realidad del problema en la facultad.

Nos entrevistamos con el Prof. Alfredo Piria, del IMERL, quien también nos dio su visión del problema.

El Prof. Marcelo Cerminara, quien estuvo trabajando más directamente en el problema de asignación, nos describió más detalladamente el problema de asignar las materias de primer año. El fue quien tuvo que realizarla, en forma manual, para el año 1998. También nos proporcionó planillas con el resultado de su trabajo, las cuales utilizamos como datos de entrada para nuestro segundo grupo de pruebas, en la sección 9 de este informe.

2.4 Clasificación Problema y Estudios realizados sobre problemas similares

El problema de 'agendar' un conjunto de items en un rango de horarios de modo de satisfacer ciertas condiciones se conoce en la literatura como 'timetable problem'. Este problema se caracteriza por manejar un gran volumen de datos y diversos tipos de restricciones.

Existe una variante del problema que deseamos resolver que ha sido bastante estudiada y a la que se han aplicado diversas técnicas para su resolución. Esta

variante considera la asignación de clases o ‘lecciones’ a horarios en la semana teniendo en cuenta la disponibilidad de los docentes y los requerimientos del curso y los estudiantes. Los salones en sí no son parte fundamental del problema, ya que en general los alumnos de un grupo permanecen en un salón fijo y los profesores se trasladan de un salón a otro para dar la lección. No se trata de un problema en que los salones sean un recurso escaso. Sin embargo, el problema tiene varias similitudes con el nuestro, como requerimientos de horario para las clases, espaciamiento de las mismas en la semana, compactibilidad de la asignación (que no existan ‘puentes’ ni para los alumnos ni para los profesores), existencia de preasignaciones, etc. Estos requerimientos surgen debido a la disponibilidad del docente o a objetivos didácticos y comodidad de los estudiantes.

En todos los estudios hechos se reconoce un problema NP-completo debido a las condiciones que la asignación debe satisfacer [8][11][12].

Las soluciones presentadas se basan en heurísticas, debido al costo de hallar una solución exacta. Principalmente se utilizan Tabú Search [12][14][15] y Algoritmos Genéticos [10][11] con buenos resultados, aunque la técnica a utilizar depende en gran medida de la variante considerada del problema.

3 Formalización del Problema

3.1 Planteo General de un Problema de Optimización

En su forma más general, un problema de optimización consiste en encontrar un elemento 'x' perteneciente a un dominio F tal que minimice el valor de una función 'f'.

El planteo matemático correspondiente es:

$$\text{mín } f(x) / x \in F.$$

donde F es el dominio considerado, llamado 'región factible' y f es conocida como 'función objetivo'.

Restringiéndonos a los problemas llamados de 'programación matemática', podemos considerar:

$$x \in R^n, f:R^n \rightarrow R, F \subseteq R^n.$$

La región F está delimitada por las restricciones del problema. La misma puede expresarse como los elementos x que satisfacen cierto conjunto de inecuaciones:

$$F = \{ x / g(x) \leq 0 \} \text{ con } x \in R^n , \ g: R^n \rightarrow R^m .$$

En forma más explícita tenemos:

$$g = (g_1, g_2, \ldots, g_m)$$
$$x = (x_1, x_2, \ldots , x_n)$$

y el conjunto de inecuaciones es:

$$g_1(x_1, x_2, \ldots, x_n) \leq 0$$
$$g_2(x_1, x_2, \ldots, x_n) \leq 0$$
$$\ldots$$
$$g_m(x_1, x_2, \ldots, x_n) \leq 0$$

Las variables x_i se llaman 'variables independientes' o 'variables decisionales'.

Acercándonos un poco más al problema que queremos resolver, podemos plantear la formulación típica de un problema de asignación: ATP (Assignment Type Problem) [7].

El ATP consiste en, dados 'n' items y 'm' recursos, determinar una asignación de los recursos a los items optimizando una función objetivo y satisfaciendo 'K' restricciones:

mín f(x)

sujeto a: $\sum_{j \in J_i} x_{ij} = 1 \quad 1 \le i \le n$ (1)

$G_k(x) \le 0 \quad 1 \le k \le K$ (2)

$x_{ij} \in \{0,1\} \quad 1 \le i \le n, \ j \in J_i$ (3)

donde:

x_{ij} son las variables de decisión: $x_{ij} = 1$ si el recurso j es asignado al ítem i
0 si no.

$J_i \subseteq \{1, 2, \ldots, m\}$ es el conjunto de recursos admisibles para el ítem 'i'.

Las restricciones (1) y (3) implican que cada ítem 'i' debe tener asignado exactamente un recurso 'j'.

La función objetivo 'f' y las restricciones G_k no necesitan cumplir ninguna propiedad en particular.

3.2 Descripción del Problema como un Problema de Optimización

Nuestro problema puede plantearse como un problema de optimización combinatoria (C.O.P.). Un COP se especifica como un conjunto de instancias de problema. Una instancia se define como un par (S, f) donde S es el conjunto finito de soluciones factibles, llamado espacio de soluciones, y f es una función de costo f: S -> R. El valor óptimo de f es

$f_0 = \text{mín} \{f(i): i \in S\}$

y el conjunto de soluciones óptimas es

$S_0 = \{i \in S: f(i) = f_0\}$

El objetivo es encontrar alguna solución $i_0 \in S_0$.
En nuestro problema la función f indica el grado de inaceptabilidad de la asignación realizada, es decir el grado de desfasaje de la asignación generada con una asignación ideal en la que se cumplen todas las restricciones y se satisfacen todas las preferencias.

3.2.1 Restricciones

Son las condiciones que debe cumplir la solución al problema para ser considerada factible. Recordar que un curso está formado por varias clases, todas distintas entre sí, dictadas cada una en un día de la semana, en iguales o distintos horarios y salones. Las restricciones a considerar son:

1. la cantidad de horas asignadas a una clase no puede ser menor a la duración de la clase.
2. una clase debe asignarse a un único salón.
3. una pareja (horario, salón) debe asignarse a una única clase.
4. los cursos deben asignarse dentro de los horarios existentes en la tabla de horarios.
5. una clase debe comenzar y finalizar dentro del mismo día.
6. los cursos de un mismo año no pueden superponerse (pueden considerarse excepciones si existen varios grupos para un curso dado, por ej. varios prácticos de análisis que se dictan a la misma hora, en distintos salones, con el solo fin de repartir a los estudiantes).
7. Contigüidad: las horas asignadas a una clase deben ser contiguas.
8. los grupos que tienen un turno especificado sólo pueden asignarse dentro de la franja horaria correspondiente al turno.
9. restricciones de horarios y días: indican que la clase no puede asignarse fuera del rango horario especificado, de lo contrario la asignación no es factible.
10. Capacidad: una clase no puede asignarse a un salón con menor capacidad de la requerida por la clase.
11. restricciones de “otros requerimientos”: si la asignación de una clase no satisface los requerimientos especificados, entonces la asignación no es factible.
12. “recursos móviles”: son aquellos que no dependen del salón de clase, sino que pueden ser trasladados de un salón a otro. Las clases no pueden asignarse de modo que no alcancen estos recursos en un horario dado.
13. no pueden dictarse dos clases de un mismo grupo un mismo día.

Algunas de estas restricciones podrían considerarse como preferencias, dependiendo de cómo se desee tratar el problema. En el momento de realizar el diseño definitivo, debe elegirse una de las dos opciones.

3.2.2 Preferencias

Las ‘preferencias’ son condiciones que es deseable que la asignación cumpla. Difieren de las restricciones en que son más ‘blandas’. Si una solución no cumple una preferencia, no deja de ser factible, mientras que si viola una restricción, dejará de serlo. Por lo tanto las preferencias formarán parte de la función objetivo.

- Horarios y/o días - La mayoría de los cursos, por diversas razones, es preferible que sean dictados en determinados horarios y o días.
- “Otros requerimientos” - Estos requerimientos corresponden a características especiales que debe tener un salón para que pueda ser dictada una clase. Por ej., si la clase necesita retroproyector, el salón debe tener enchufe.
- no superposición entre cursos de distintos años. - Para controlar esta preferencia se definirán entidades que llamaremos ‘**perfiles**’, las cuales agruparán un conjunto de cursos (o clases) que no es deseable que se superpongan, dado que estadísticamente se sabe que un gran número de estudiantes los cursan a la vez.
- adyacencia (entre teóricos y prácticos, o materias de un mismo año) - Es deseable que los teóricos y los prácticos correspondientes se dicten en horarios contiguos, esto también es válido para materias de un mismo año.
- mismo salón (algunos teóricos y prácticos, materias de un mismo año). - Por ej. para materias de un mismo año que son cursadas por gran cantidad de estudiantes, puede resultar bueno que se dicten en el mismo salón para evitar traslados masivos de los estudiantes.
- capacidad del salón - No sería bueno que se utilizaran salones de gran capacidad para materias de pocos alumnos.
- espaciamiento en los días de la semana de las clases de un mismo curso/grupo. - Por motivos didácticos es conveniente que las clases de un grupo se repartan en forma uniforme en la semana.
- igualdad de horarios en las distintas clases de un grupo.
- compactibilidad: los cursos de un año deben asignarse evitando que existan “puentes” entre una clase y otra.

La preferencia de un rango horario ayuda a establecer automáticamente otras preferencias como no superposición, adyacencia o espaciamiento.
Por ej. si quiero que los prácticos se asignen a continuación de los teóricos, ingresaré como datos rangos de horarios adecuados para cada curso.

Existen preferencias que tendrán prioridad respecto a otras, ya que algunas podrían resultar contradictorias entre sí, o contradictorias respecto a alguna restricción. Por ejemplo, no puede exigirse el espaciamiento de las clases en la semana si se especificó como preferencia horaria dos días específicos consecutivos.

3.2.3 Función Objetivo

Para evaluar la conveniencia de la asignación realizada, definimos una función que dada la asignación devolverá un valor que será menor cuanto más cerca de la asignación ideal se encuentre. La asignación ideal es aquella que satisface totalmente las restricciones, y tendría un valor cero según la función objetivo.

La misma tendrá dos componentes principales:

- minimizar la cantidad de elementos (grupos y/o clases) no asignados.

  Considerando los datos de la realidad, es poco probable que no lleguen a asignarse todos los cursos de un año, o la totalidad de un año o carrera, pero sí podría ocurrir que alguna clase, grupo o curso quede sin asignar.

  El hecho de que exista alguna asignación sin realizar es un problema más grave que el de una asignación que no satisfaga plenamente el resto de las preferencias, y se evaluará con una penalización mayor; de ahí el interés de considerarlo como un elemento aparte.

  Pueden asignarse distintos pesos a cada elemento, ya que por ejemplo es más grave que no se haya asignado un curso, que no se haya asignado alguna clase de un curso. En el primer caso, el curso no podrá dictarse, mientras que en el segundo tendrá menos carga horaria que la requerida.

- minimizar desfasajes con las preferencias.

  Dada una preferencia, existirá un componente de la función objetivo que la representará. El valor de este componente será cero si la preferencia es satisfecha por la solución, y mayor que cero si no. El valor será tanto mayor cuanto más nos alejemos del cumplimiento de la preferencia. Por ejemplo, si una clase con determinada preferencia de horario se asigna media hora más tarde, se penalizará menos que si se asigna tres horas más tarde. Por otro lado, distintos tipos de preferencias podrán tener distinto ‘peso’, es decir, podrán ser ponderadas por factores de distintos órdenes.

3.3 Formalización

3.3.1 Definición de Entidades

Definimos las siguientes entidades:

- S = conj. de salones.
- H = conj. de horarios (intervalos de ½ hora en la semana). Se considerarán intervalos discretos de tiempo. Dadas las características de los cursos existentes en la Facultad de Ingeniería, puede considerarse un intervalo mínimo de media hora. De este modo podrán representarse clases de distinta duración.
- C = conjunto de clases
- G = conjunto de grupos = $\{g_1, g_2, ..., g_n\}$ / $g_i \subseteq P(C)$, donde P(C) es el conjunto de partes de C (conjunto potencia).
- CURSOS = conjunto de cursos = $\{c_1, c_2, ... , c_m\}$ / $c_i \subseteq P(G)$.
- AÑOS = conj. de años: = $\{ a_1, a_2, ... , a_j \}$ / $a_i \subseteq P(CURSOS)$.
- CARRERAS = conj. de carreras = $\{cr_1, cr_2, ... , cr_k\}$ / $cr_i \subseteq P(AÑOS)$.

- R = tipos de recursos móviles
- T = conjunto de turnos = {t / t es un rango de horarios}
- AÑO-TURNO = { (a, t) / a $\in$ AÑOS, t $\in$ TURNOS }
- PERFILES = { (c,e) / c $\in$ *P*(CURSOS), e $\in$ ENTEROS } donde e es el numero de estudiantes del perfil.

- CARR_AÑOS = { (id_carrera, id_año, conj_cursos) / id_carrera $\in$ CARRERAS, id_año $\in$ AÑOS, conj_cursos $\in$ *P*(CURSOS) }

  'conj_cursos' es el conjunto de cursos correspondientes a un año de una carrera.
- clase = (id_clase, rangos horarios de pref., cant. de alumnos, importancia, turno, duración, otros_requerimientos)

  Los 'rangos horarios de preferencia' son un conjunto de parejas (horario de comienzo, horario de fin) dentro del rango de horas de la semana, de modo que para cada clase pueden especificarse varios rangos de preferencia.

Definimos una matriz **A** de tres dimensiones que representará las asignaciones realizadas:

$A_{m \times n \times p}$ (salones $\times$ horarios $\times$ clases)

Sus elementos son de la forma:

a_{shc} = 1 si la clase "c" se asignó al salón "s" a la hora "h"
0 en otro caso.

obs.: la hora "h" indica un intervalo de ½ hora.

3.3.2 Restricciones

Nota: En la siguiente formalización se define el operador "[ ]" como una función que aplicada a una expresión devuelve un booleano (0 o 1).

[expr] = 1 si expr es verdadero
0 si expr es falso.

1. la cantidad de horas asignadas a una clase no puede ser menor a la duración de la clase.

$$\sum_{h \in H} \sum_{s \in S} a_{shc} = \text{duración}(c) \quad \forall c \in C$$

2. una clase debe asignarse a un único salón.

$$\sum_{s \in S} [\ (\sum_{h \in H} a_{shc}) \geq 1 \ ] \leq 1 \qquad \forall c \in C.$$

Esta restricción indica que las horas asignadas a una misma clase deben corresponder a un solo salón, es decir, no puede asignarse parte de una clase a un salón y parte a otro.

3. una pareja (horario, salón) debe asignarse a una única clase.

$$\sum_{c \in C} a_{shc} \leq 1 \quad \forall s \in S, \forall h \in H$$

4. los cursos deben asignarse dentro de los horarios existentes en la tabla de horarios.

   Se cumple por la formalización elegida.

5. una clase debe comenzar y finalizar dentro del mismo día.

   Sea p la cantidad de ½ horas en un día.
   Asumiendo que la primer ½ hora del día es la hora cero:

$$\sum_{0 \leq h \leq (\#H - 1)} a_{shc} * a_{s(h+1)c} * [\ \lfloor h/p \rfloor \neq \lfloor (h+1)/p \rfloor \ ] = 0 \quad \forall s \in S, \forall c \in C.$$

   Esta restricción recorre la matriz de asignación verificando que los a_{shc} contiguos que están en 1 corresponden al mismo día de clases. Para que esta restricción pueda controlarse correctamente, debe satisfacerse la restricción 7.

6. las clases de los cursos de un mismo año no pueden superponerse (pueden considerarse excepciones si existen varios grupos para un curso dado, por ej. varios prácticos de análisis que se dictan a la misma hora, en distintos salones, con el solo fin de repartir a los estudiantes).

$$\sum_{cur \in Conj\_cursos(ca)} \sum_{c \in cur} \sum_{s \in S} a_{shc} \leq 1 \quad \forall h \in H, \forall ca \in CARR\_AÑOS.$$

7. Contigüidad: las horas asignadas a una clase deben ser contiguas.

$$\sum_{1 \leq h \leq (\#H - 1)} [a_{shc} \neq a_{s(h+1)c}] \leq 2 \qquad \forall s \in S, \forall c \in C.$$

   Esta restricción cuenta el número de cambios 0-1 y 1-0 en la matriz de asignación. Para que las ½ horas asignadas sean contiguas, el nro. de cambios debe ser como máximo 2.

8. los grupos que tienen un turno especificado solo pueden asignarse dentro de la franja horaria correspondiente al turno.

$$\sum_{s \in S} \sum_{h \notin turno(c)} a_{shc} = 0 \qquad \forall c \in C$$

   Un turno se define como un rango horario.

9. restricciones de horarios y días: indican que el curso no puede asignarse fuera de sus horarios de preferencia, de lo contrario la asignación no es factible.

$$\sum_{s} \sum_{h \notin hor\_pref(c)} a_{shc} = 0 \quad \forall c \in C$$

   Donde hor_pref(c) indica los horarios de preferencia de la clase ‘c’.

10. Capacidad: una clase no puede asignarse a un salón con menor capacidad de la requerida por la clase.

$$\sum_{s \in S} \sum_{h \in H} \sum_{c \in C} a_{shc} * [\, capacidad(s) < cant\text{-}estud(c) \,] = 0$$

    o también:

$$a_{shc} * (capacidad(s) - cant\text{-}estud(c)) \geq 0 \quad \forall s \in S, h \in H, c \in C.$$

11. Restricciones de “otros requerimientos”: si la asignación de una clase no satisface los requerimientos especificados, entonces la asignación no es factible.

$$\sum_{s \in S} \sum_{h \in H} \sum_{c \in C} a_{shc} * [\ otros\_req(c) <> otros\_req(s)\ ] = 0$$

12. “recursos móviles”: son aquellos que no dependen del salón de clase, sino que pueden ser trasladados de un salón a otro. Las clases no pueden asignarse de modo que no alcancen estos recursos en un horario dado.

$$\sum_{s \in S} \sum_{c \in C} a_{shc} * Recursos\_móv(c, R_i) \leq \#R_i$$

$$\forall h \in H, \forall Ri \in R$$

Donde “Recursos_mov(c, R_i)” indica la cantidad del recurso móvil de tipo R_i que utiliza la clase ‘c’

13. no pueden dictarse dos clases de un mismo grupo un mismo día.

$\forall$ curso, $\forall$ grupo $\in$ curso, $\forall$ clase c1, c2 $\in$ grupo,
$\lfloor$ hor_ini(c1)/p $\rfloor \neq \lfloor$ hor_ini(c2)/p $\rfloor$.

Nota: Se utilizan funciones auxiliares como ‘otros_req, turno, capacidad, etc. no especificadas que permiten seleccionar un elemento de una tupla, o sea un atributo de una entidad.

3.3.3 Preferencias

A continuación se expresan las preferencias formalmente.

1. Horarios y/o días

Pueden expresarse preferencias de horarios y/o días a nivel de curso, grupo o clase individual. Las ingresadas a nivel de curso o grupo deberán procesarse para llevarlas a nivel de cada clase individual. Estas preferencias pueden ser variadas, por ejemplo, podrían ser un día de la semana, algunos días, un horario en cualquier día de la semana, un día y hora determinados, etc.
Todas estas preferencias deben traducirse a uno o más rangos de medias horas y se almacenarán en un vector de rangos horarios de preferencia de la clase.
Los rangos horarios deberán ser mayores o iguales a la duración de la clase.

Para controlar el cumplimiento de esta preferencia, medimos el grado de coincidencia entre el horario asignado a la clase y sus rangos horarios de preferencia.

Para ello es necesaria una matriz que especifique los rangos horarios de preferencia de cada clase. Sea P esta matriz, $P_{c \times h}$ (clases $\times$ horarios), donde $p_{ch} = 1$ indica que el horario h es preferido por la clase c.

$$\text{Costo} = \sum_{c \in C} \sum_{h \in H} [\, [\neg\, p_{ch}] \wedge \sum_{s \in S} a_{shc} \,]$$

2. Adyacencia horaria de clases:

Esta preferencia controla que ciertas clases se asignen en horarios adyacentes, es decir, una a continuación de la otra. Puede aplicarse a clases teóricas y prácticas de un mismo grupo, o a cualquier conjunto de clases que se desee.

Si no se dan las condiciones de preferencia de horarios y/o turnos adecuadas para poder exigir la adyacencia, se considera que la misma se cumple, es decir, no se penalizará la asignación dada. Por ejemplo, si en un grupo existe una única clase de teórico y una única clase de práctico, y tienen distintos días como preferencia horaria, no es correcto pedir la adyacencia de estas dos clases.

Al aplicar esta preferencia a clases teóricas y prácticas, si la cantidad de clases teóricas es mayor que la de clases prácticas, se controlará que todo práctico sea adyacente a algún teórico, mientras que si existen más clases prácticas que teóricas, se controlará que todo teórico sea adyacente a algún práctico.

Para expresar esta preferencia utilizamos una matriz P que indica entre qué clases sería deseable la adyacencia horaria, $P_{c \times c}$ (clases $\times$ clases)

$$\text{costo} = \sum_{c1 < c2 \in P} [\, p_{c1\,c2} \wedge \neg\, [\sum_{h \in H} \sum_{s \in S} (a_{shc1}) \wedge (\sum_{s \in S} a_{s(h+1)c2}) \,] \,]$$

Obs.: se supone que c1 y c2 no quedaron superpuestas en la asignación.

3. Mismo Salón

La preferencia de que ciertas clases se dicten en el mismo salón puede expresarse en forma similar a la adyacencia horaria.

Consideramos una matriz $P_{c \times c}$ que indica qué clases deberían dictarse en el mismo salón.

$$\text{costo} = \sum_{c1 < c2 \in P} [\, p_{c1\,c2} \wedge \neg\, [\sum_{s \in S} \sum_{h1,h2 \in H} [a_{sh1c1} \wedge a_{sh2c2} \,] \,] \,]$$

Obs.: se supone que c_1 y c_2 no quedaron superpuestas en la asignación.

Aplicar preferencia no siempre es conveniente, por ejemplo en el caso de exigir el mismo salón para la clase teórica y práctica de un curso, ya que puede dar lugar a que se sacrifique un salón de alta capacidad, que sería adecuado para asignar una clase teórica, al asignarlo a una clase de práctico solamente para que la misma se dicte en el mismo salón que el teórico. Por lo tanto debe tener un peso bajo en la función objetivo, y debe ser considerada si la cantidad de estudiantes del práctico es aproximadamente igual a la de teórico.

Por otra parte, esta preferencia solo es exigible si la clase de teórico y la de práctico consideradas fueron asignadas en horarios adyacentes; por lo tanto puede controlarse junto con la preferencia de adyacencia horaria, en el mismo algoritmo.

4. Compactibilidad

Es deseable que no existan 'puentes' entre las clases de un mismo año, o sea que la asignación sea lo más 'compacta' posible dentro de un día de clases.

Sean: d = día de la semana, $0 \leq d \leq 4$
p = cantidad de horarios (1/2 horas) en un día.
AC = matriz que en sus filas tiene los años de las carreras de la facultad, y en sus columnas las clases.
ac_{ij} = 1 si la clase j $\in$ año i.
0 si no.

$$\sum_{0 \leq d \leq 4} \sum_{d*p \leq h < (d+1)*p} [(\sum_{s \in S} \sum_{c \in AC} a_{shc}) \neq (\sum_{s \in S} \sum_{c \in AC} a_{s(h+1)c})] = \text{puentes}$$

Esta expresión totaliza para todos los salones y clases de un año los $a_{shc} = 1$ y luego cuenta la cantidad de cambios 0-1 y 1-0 existentes en el vector resultante, que corresponden a la cantidad de 'puentes' en un día. Este resultado se acumula para cada día 'd' de la semana, y luego debe sumarse para todas las carreras y años.

5. Otros requerimientos

Esta preferencia puede controlarse recorriendo la matriz de asignación y viendo si los valores de estos atributos coinciden para cada salón y clase donde $a_{shc} = 1$.
Se requiere una estructura auxiliar que describa los requerimientos de cada clase y otra para las características que posee cada salón.
Sean: $OTR_{c \times r}$ (clases $\times$ requerimientos) la matriz que describe los requerimientos de cada clase
$OTC_{s \times r}$ (salones $\times$ requerimientos) la matriz que describe las características de cada salón.

$$\text{costo} = \sum_{s \in S} \sum_{h \in H} \sum_{c \in C} \sum_{r=1}^{10} [a_{shc} \wedge OTR_{cr} \wedge \neg OTC_{sr}]$$

6. Capacidad del salón

Esta preferencia relaciona la cantidad de estudiantes de una clase con la capacidad del salón a la que fue asignada.

Sean: CES_c vector de clases que indica la cantidad de estudiantes de cada una.
CAP_s vector de salones que indica la capacidad de cada uno.

$$costo = \sum_{s \in S} \sum_{h \in H} \sum_{c \in C} [\, a_{shc} \wedge [\, CES_c > CAP_s \,] \,]$$

7. Desperdicio del salón

Es análoga a la anterior. Podría refinarse esta preferencia considerando que existe desperdicio de la capacidad de un salón si el mismo es superior a un 30%. Por ejemplo:

$$costo = \sum_{s \in S} \sum_{h \in H} \sum_{c \in C} [\, a_{shc} \wedge [\, CES_c < (CAP_s * 70\%) \,] \,]$$

8. Igualdad en los horarios de comienzo de las clases teóricas de un mismo grupo

Esta preferencia se apoya en la restricción de que no pueden existir dos clases teóricas de un mismo grupo el mismo día.

Consideramos los siguientes conjuntos:

Grupos = $\{ g_1, g_2, ... / g_i \in P(Clases) \}$ conjunto de todos los grupos existentes.
Clases = $\{ c_1, c_2, ... \}$

Sea d = día de la semana, $0 \leq d \leq 4$.
p = cantidad de horas en un día.

$$costo = \sum_{g} \sum_{\substack{c1<c2 \\ c1,\, c2 \in g}} \sum_{0 \leq h < p}^{4} [\, [\, \neg \sum_{d=0}^{4} \sum_{s} a_{s\,(h+p*d)\,c1} \,] \wedge [\, \sum_{d=0}^{4} \sum_{s} a_{s\,(h+1+p*d)\,c1} \,] \wedge [\neg \sum_{d=0}^{4} \sum_{s} a_{s\,(h+1+p*d)\,c2} \,]]$$

$$\vee [\, [\, \sum_{d=0}^{4} \sum_{s} a_{s\,(p*d)\,c1} \,] \wedge [\, \neg \sum_{d=0}^{4} \sum_{s} a_{s\,(p*d)\,c2} \,] \,]$$

Se consideran las clases de cada grupo g en forma ordenada (c1 < c2) y se controlan 2 a 2 que comiencen a la misma hora del día.
La expresión considera dos casos: El primer operando del "$\vee$" contempla el caso general en que las clases no tienen asignados horarios en la primer hora del día. El segundo operando considera el caso en que las clases comienzan en la primer hora del día.

9. No superposición entre materias pertenecientes a un perfil

Sea un perfil P = {cur / cur ∈ Cursos}

$$\sum_{h \in H} \left(\left(\sum_{cur \in P} \sum_{g \in cur} \sum_{c \in G} \sum_{s \in S} a_{shc} \right) - 1 \right) = cant\_superp$$

superposiciones = peso(perfil) * cant_superp

donde peso es un atributo de perfil que indica su importancia. A mayor peso, mayor importancia.

10. Espaciamiento en días de las clases teóricas de un grupo

Esta preferencia permite expresar que es conveniente que las clases de un mismo grupo se asignen a días distantes de la semana. Por ejemplo, si un curso tiene 3 clases por semana, no es conveniente que éstas se asignen a 3 días consecutivos. Para controlar esta preferencia deben tenerse en cuenta las preferencias de horarios ingresadas.

Una forma de evaluar esta preferencia es medir la distancia en días entre la primera y la última clase de un grupo. Dependiendo de la cantidad de clases por semana, existirá un mínimo exigible para esta 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: no aplicable.

A continuación se muestran algunos ejemplos de asignación y la distancia entre la primera y la última clase en cada caso.

- 2 clases por semana:

L	X
M	
M	X
J	
V	

2

L	X
M	
M	
J	
V	X

4

L	X
M	X
M	
J	
V	

1

- 3 clases por semana:

L	X
M	
M	X
J	
V	X

4

L	X
M	
M	X
J	X
V	

3

L	X
M	X
M	X
J	
V	

2

- 4 clases por semana:

L	X
M	X
M	X
J	
V	X

4

L	X
M	X
M	X
J	X
V	

3

Sean:

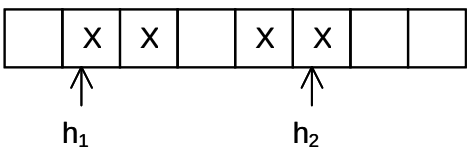
ccg = cantidad de clases de un grupo g = $\sum_{c \in g}$ teórico(c)

H = cantidad de horarios existentes.
DIA $_h$ = vector de horarios que indica a qué día corresponde el horario.
DIST $_{ccg}$ = vector que indica dada la cantidad de clases de un grupo (ccg) qué distancia mínima en días deben tener, entre la primera clase que se dicte el la semana y la última.
X $_h$ = vector booleano de horarios que indica los horarios ocupados por las clases de un grupo:

$$x_h = \sum_{c \in g} \sum_{s} (\text{teórico}(c) \wedge a_{shc})$$

Para controlar el espaciamiento, se recorre el vector X desde el comienzo hasta encontrar la primera hora ocupada h_1 y desde el final hasta encontrar la última h_2. Luego se obtienen los días a que corresponden h_1 y h_2 por medio del vector DIA. Si la diferencia entre estos 2 días es menor a la especificada en el vector DIST para la cantidad de clases del grupo (ccg), entonces se acumula el costo de no satisfacer esta preferencia.

vector X:



$$\text{costo} = \sum_{g \in G} \sum_{h1=0}^{H-1} \sum_{h2=0}^{H-1} \sum_{i=0}^{h1} [\, [\, \Sigma\, x_i = 0\,] \wedge x_{h1+1}\,] \vee [\, x_{h1} \wedge \neg\, h_1\,]\,]$$

$$\wedge [\, [\, [\, \sum_{j=0}^{h2} x_{(H-1-j)} = 0\,] \wedge [\, x_{(H-1-h2-1)} = 1\,]\,] \vee [\, x_{h2} \wedge h_2 = H-1\,]\,]$$

$$\wedge [\, DIA_{(H-1-h2-1)} - DIA_{(h1+1)} < DIST_{(ccg)}\,]$$

3.3.4 Función Objetivo

La función objetivo estará compuesta por un término que expresará el grado de insatisfacción de cada una de las preferencias especificadas en el punto anterior, ponderadas según su importancia. Podrá tener además un término que indique la cantidad de clases sin asignar.

$fo = peso_1 * costo_1 + peso_2 * costo_2 + ... + peso_n * costo_n$

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

3.4 Complejidad de Nuestro Problema

3.4.1 Problemas NP- Completos

Hemos expresado nuestro problema como un COP, problema de optimización combinatoria, el cual corresponde a lo que comúnmente se llama problema de búsqueda. Existe una clase de problemas más simple, los problemas de decisión que tienen la siguiente forma general:

Dado un espacio de soluciones S, una función de costo f, una cota θ, ¿existe alguna solución factible $i \in S$ tal que $f(i) \leq \theta$?

Es evidente que si es posible resolver un COP de búsqueda entonces uno puede resolver la correspondiente versión de decisión, mientras que el recíproco no necesariamente se cumple.

Sea n el largo (i.e. el número de dígitos binarios) de la codificación de una instancia del COP. Si la cantidad máxima de tiempo de cálculo necesario para resolver una instancia de largo n está acotada superiormente por un polinomio en n, para cada n, diremos que el COP en estudio es resoluble en tiempo polinómico y que el algoritmo

solución correspondiente es polinómico. Si k es el máximo exponente de tal polinomio diremos que el COP es resoluble en un tiempo de orden n^k ($O(n^k)$).

P denota la clase de problemas de decisión para los cuales existe un algoritmo que determine en tiempo polinómico para cada instancia si la respuesta es 'sí' o 'no'.
NP denota la clase de problemas de decisión para los cuales existe un algoritmo que verifique en tiempo polinómico para cada instancia si la respuesta 'sí' es correcta.
Obviamente $P \subseteq NP$ y el problema de encontrar si esta inclusión es propia o no, es uno de los problemas abiertos más importantes de la matemática actual.
La mayoría de los problemas combinatorios de decisión están en NP mientras que sólo unos pocos de ellos, aunque en la práctica muy relevantes, pertenecen a P.

Algunos de los más importantes COPs son de hecho NP-completos, es decir tan 'difíciles' de resolver como el problema más complejo en NP. Los métodos exactos para resolver problemas conocidos de esta clase requieren de un número exponencial de pasos: entonces cuando la dimensión n de la instancia se hace más y más grande como ocurre en la mayoría de las aplicaciones de la vida real, ningún algoritmo exponencial podría ser de utilidad práctica.

3.4.2 Problema de Asignación Cuadrático (QAP - Quadratic Assignment Problem)

Descripción y Formalización

El QAP de orden n consiste en encontrar la mejor asignación de n servicios a n localidades, considerados los términos 'servicio' y 'localidad' en su sentido más amplio. En general, la formalización matemática del mismo [6] es la siguiente:

Se definen tres matrices de dimensión n x n:

$D = [d_{ih}]$ = matriz de distancias entre las localidades (entre localidad i y localidad h).
$F = [f_{jk}]$ = matriz de flujos entre los servicios (entre servicio j y servicio k).
$C = [c_{ij}]$ = matriz de costos de la asignación de un servicio a una localidad (de servicio j a localidad i).

Por 'flujo' se entiende grado de dependencia, relacionamiento o intercambio entre dos servicios, ya que podría considerarse que los distintos servicios no son totalmente independientes entre sí, y la forma como sean asignados a las localidades afecta el resultado final.

Normalmente las matrices D y F son matrices simétricas de valores enteros, mientras que el costo c_{ij} de asignar el servicio j a la localidad i es usualmente ignorado, ya que no contribuye significativamente a la complejidad del problema.

Sean por ejemplo:

$$D = \begin{vmatrix} 0 & 1 & 1 & 2 & 3 \\ 1 & 0 & 2 & 1 & 2 \\ 1 & 2 & 0 & 1 & 2 \\ 2 & 1 & 1 & 0 & 1 \\ 3 & 2 & 2 & 1 & 0 \end{vmatrix} \qquad F = \begin{vmatrix} 0 & 5 & 2 & 4 & 1 \\ 5 & 0 & 3 & 0 & 2 \\ 2 & 3 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 5 \\ 1 & 2 & 0 & 5 & 0 \end{vmatrix}$$

La asignación de cada servicio a cada localidad puede verse como una permutación de servicios Π, dadas las localidades en cierto orden fijo:

Π: loc $\rightarrow$ serv / Π(l) = s, si el servicio 's' ha sido asignado a la localidad 'l'.

donde: loc = conjunto de localidades
serv = conjunto de servicios
l $\in$ loc, s $\in$ serv.

Esta permutación debe minimizar el costo total de la asignación. El costo de transferir datos, o materiales, etc. entre dos servicios (o sea flujo entre los servicios) puede expresarse como el producto de la distancia entre las localidades a las que fueron asignados los servicios y la cantidad de flujo entre los mismos:

$$d_{ij} \cdot f_{\pi(i)\,\pi(h)}$$

donde d_{ih} es la distancia entre las localidades 'i' y 'h'
y $f_{\pi(i)\,\pi(h)}$ es la cantidad de flujo entre los servicios π(i) (servicio asignado a la localidad i) y π(h) (servicio asignado a la localidad h).

Por lo tanto la minimización del costo de la asignación de servicios a localidades puede expresarse por medio de la siguiente función objetivo **z**, la cual incluye además el costo de asignar un servicio a una localidad, $c_{i\pi(i)}$:

$$\text{mín } \mathbf{z} = \sum_{i,h=1}^{n} d_{ih} \cdot f_{\pi(i)\,\pi(h)} + \sum_{i=1}^{n} c_{i\pi(i)}$$

Naturaleza Cuadrática del problema:

Podemos reformular el problema, de modo que resolverlo implique encontrar una matriz $A_{n\times n}$ cuyas filas son servicios y cuyas columnas son localidades, y cuyos elementos a_{ih} son 1 si el servicio 'i' fue asignado a la localidad 'h' y 0 en otro caso, podemos ver que la función objetivo es cuadrática:

$$\text{mín } \mathbf{z} = \sum_{i,j=1}^{n} \sum_{h,k=1}^{n} d_{ih} \cdot fj_{k} \cdot a_{ij} \cdot a_{hk} + \sum_{i,j=1}^{n} c_{ij} \cdot a_{ij}$$

donde los elementos a_{ij} deben cumplir las siguientes restricciones:

- $\sum_{i=1}^{n} a_{ij} = 1 \qquad (j = 1 \ldots n)$

- $\sum_{j=1}^{n} a_{ij} = 1 \qquad (i = 1 \ldots n)$

- $a_{ij} \in \{0,1\} \qquad (i, j = 1 \ldots n)$

Esto se debe a que existen relaciones entre las localidades (distancias) y entre los servicios (flujos), además del costo de cada asignación particular (dado por 'c_{ij}').

3.4.3 Analogía con el QAP

Nuestro problema puede identificarse fácilmente con el problema de asignación cuadrático (QAP), ya que se trata de asignar salones y horarios a clases. La diferencia principal consiste en que nuestro problema consta de tres dimensiones básicas (salones, horarios y clases) en lugar de dos (servicios y localidades) .

Las tres dimensiones pueden reducirse a dos agrupando dos de ellas.

3.4.4 Naturaleza Cuadrática de la función objetivo del Problema de Asignación

Demostraremos que la función objetivo de nuestro problema es al menos cuadrática. Para ello asumiremos algunas hipótesis restrictivas que nos permitirán simplificar el problema de modo de expresarlo en forma idéntica al Q.A.P. Sabemos que este último tiene una función objetivo cuadrática. Por lo tanto, nuestro problema será al menos tan complejo como el Q.A.P.

Consideraremos para la demostración la preferencia de 'adyacencia de clases', de modo que nuestra función objetivo será minimizar la cantidad de clases que no cumplan este requisito. Asumiremos también que:

1. tenemos igual número de clases y horarios: |C| = |H| = n.
2. las clases tienen todas una duración de un intervalo horario.
3. existe un único salón al cual pueden asignarse las clases.

Las preferencias de adyacencia estarán dadas por una matriz $P_{n \times n}$ donde:

$$p_{ih} = \begin{cases} 1 \text{ si la clase i debe ser adyacente a la clase h } (i \neq h) \\ 0 \text{ si no} \end{cases}$$

Consideramos una función auxiliar, también en forma de matriz $H_{n\times n}$ que nos indica horarios contiguos:

$$h_{jk} = \begin{cases} 1 \text{ si } k = j + 1 \\ 0 \text{ si no.} \end{cases}$$

Ya que consideramos un solo salón, nuestra matriz de asignación será $A_{n\times n}$ donde:

$$a_{ij} = \begin{cases} 1 \text{ si la clase i se asignó al horario j.} \\ 0 \text{ si no.} \end{cases}$$

A cumple las siguientes restricciones:

1. $\sum_{c}^{n} a_{ij} = 1 \quad \forall h$

2. $\sum_{h}^{n} a_{ij} = 1 \quad \forall c$

3. $a_{ij} \in \{0,1\} \quad \forall i, j$

La matriz (- P) se corresponde con la matriz de distancias D del Q.A.P., y la H con la matriz de flujos F.

De este modo estamos bajo las hipótesis del Q.A.P., y planteamos la función objetivo como:

$$\text{mín } z = \sum_{i,j}^{n} \sum_{h,k}^{n} - p_{ih} \cdot h_{jk} \cdot a_{ij} \cdot a_{hk}$$

la cual es análoga a la función del Q.A.P. La misma tomará un valor menor (más negativo) cuanto más adyacencias se cumplan. Se considera que la matriz C es nula (costos de asignación despreciables).

Como el Q.A.P. es una generalización del T.S.P., y éste es un problema NP-completo, entonces el Q.A.P. también lo es [5]. Dado que, como acabamos de ver, nuestro problema es al menos tan complejo como el Q.A.P. , se deduce que es también NP-completo.

4 Estudio de Técnicas y su Posible Aplicación al Problema

4.1 Técnicas Exactas

Estas técnicas se basan en la formulación del problema como problema de programación matemática y buscan una solución exacta al mismo.

Gams

Es una herramienta desarrollada para resolver problemas de programación matemática [16].

Permite la representación compacta de modelos grandes y complejos a través de un lenguaje de alto nivel. El modelo del problema puede describirse en forma independiente de los algoritmos que se utilicen para solucionarlo y de los datos de una instancia particular del problema.

El diseño de GAMS incorpora ideas de la teoría de bases de datos relacionales y de la programación matemática para cubrir las necesidades de quienes modelan el problema. Las bases de datos relacionales permiten una mejor organización y posibilidad de transformación de los datos. La programación matemática provee una forma de describir el problema y brinda una variedad de métodos para resolverlo.

Se diseñó teniendo en cuenta que:

- Todos los métodos algorítmicos existentes deberían poder ser utilizados sin necesidad de cambiar la representación del problema. También es posible la introducción de nuevos métodos o nuevas implementaciones de métodos ya existentes sin alterar los modelos existentes.
- El problema de optimización se expresa independientemente de los datos que utiliza. La separación de la lógica y los datos permite considerar problemas de diversos tamaños sin causar un aumento de la complejidad de la representación.
- El uso de bases de datos relacionales evita complicaciones al momento de representar los datos.

Aplicación al problema:

Debería expresarse el problema formalmente, como un problema de programación matemática. Dado que nuestro problema tiene gran número de restricciones y una función objetivo al menos cuadrática, no es posible estar seguro de si GAMS, aplicando las técnicas que tiene incorporadas, aportará una buena solución en un tiempo razonable.

4.2 Heurísticas

A continuación analizamos las siguientes heurísticas y la posibilidad de su aplicación al problema a resolver:

1. AS - Ant Systems
2. GA - Genetic Algorithms
3. SA - Simulated Annealing
4. SC - Sampling and Clustering
5. TS - Tabu Search
6. NN - Neural Nets

Estas heurísticas derivan de la observación de fenómenos de la naturaleza. Se inspiran en algunos casos en la biología, en otros en la física, las ciencias sociales. Se caracterizan por:

- utilizar un cierto numero de ensayos repetidos
- emplear uno o más agentes (neuronas, cromosomas, hormigas).
- operar con un mecanismo de competición-cooperación (en el caso de más de un agente)
- tener embebidos procedimientos de automodificación de los parámetros de la heurística o de la representación del problema.

4.2.1 Ant Systems - Colonias de Agentes Cooperativos

4.2.1.1 Descripción

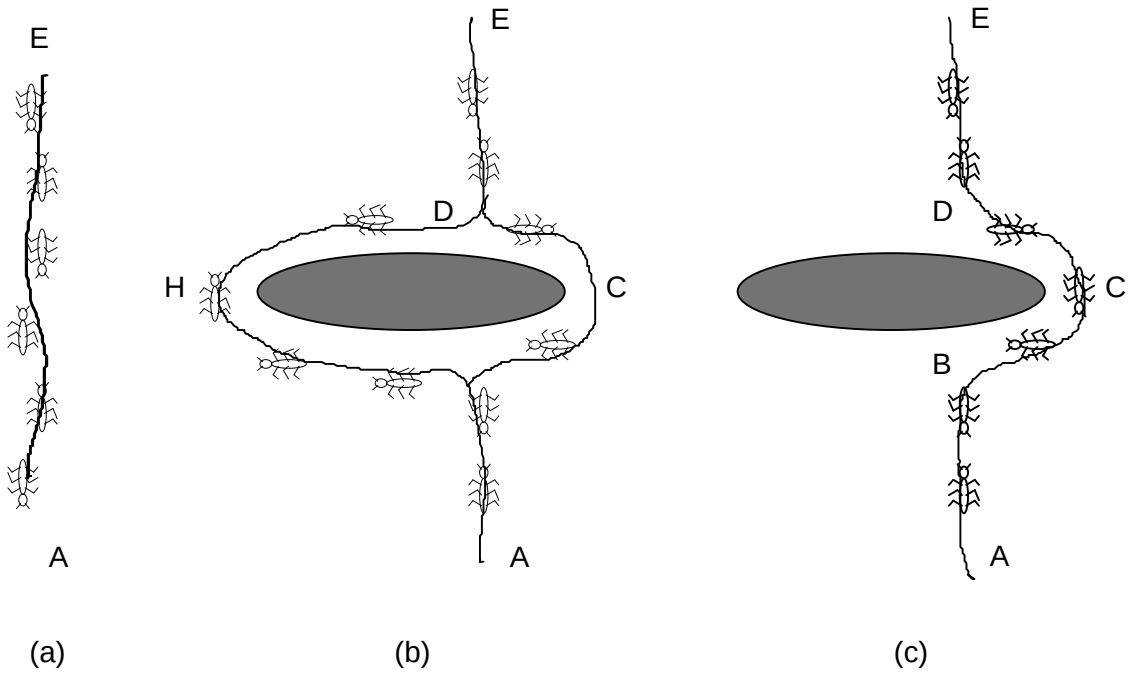
Esta heurística se basa en imitar el comportamiento de las colonias de hormigas. La colonia en su conjunto logra un comportamiento que podríamos llamar inteligente, a pesar de que cada hormiga considerada individualmente tiene solamente capacidades básicas. Este es el resultado de interacciones coordinadas entre las mismas, es decir, existe una cooperación entre agentes simples para llegar a un objetivo.

Cada hormiga realiza acciones simples, sin saber explícitamente lo que las demás están haciendo. Las posibilidades de comunicación entre hormigas son muy limitadas, por lo tanto las interacciones deben basarse en flujos simples de información. Sin embargo, a pesar de ser insectos casi ciegos, logran encontrar el camino más corto entre el hormiguero y una fuente de comida y luego regresar. Estudios etológicos han demostrado que esta capacidad es el resultado de la interacción vía comunicación química entre las hormigas y un fenómeno emergente causado por la presencia simultánea de muchas hormigas.

La comunicación entre ellas se da por medio de una sustancia química llamada feromona. Esta sustancia va siendo depositada a lo largo del camino recorrido por la hormiga, y este 'rastro' sirve de referencia a otras hormigas, indicando caminos seguidos anteriormente. Una hormiga aislada se mueve esencialmente al azar, pero si

encuentra una huella dejada previamente, es altamente probable que la siga, reforzándola con su propia feromona. El comportamiento colectivo resultante es 'autocatalítico', cuanto más hormigas siguen una huella, más atractiva se torna la misma de ser seguida. El proceso entonces se caracteriza por un loop de retroalimentación positiva , donde la probabilidad con que una hormiga elige un camino aumenta con el número de hormigas que previamente eligieron el mismo camino.

Veamos un ejemplo:



(a) (b) (c)

En la figura a) tenemos un camino seguido por las hormigas desde la fuente de comida A al hormiguero E, ida y vuelta. Si repentinamente colocamos un obstáculo cortando el camino, las hormigas intentarán rodearlo, algunas irán inicialmente por la izquierda y otras por la derecha (figura b). La primera en llegar al obstáculo (punto B o D) tiene la misma probabilidad de doblar a la derecha o a la izquierda, ya que no había previamente feromona en ninguno de los caminos alternativos. Debido a que el camino BCD es más corto que el camino BHD, la primer hormiga siguiendo el BCD llegará antes que aquella que siga el camino BHD. El resultado es que una hormiga que esté volviendo desde E hasta D encontrará una huella más fuerte en el camino DCB causada por la mitad de todas las hormigas que decidieron recorrer el obstáculo vía DCBA y por las que ya llegaron vía BCD. Por lo tanto ahora la probabilidad de elegir el camino DCB será mayor que la del camino DHB. Como consecuencia la cantidad de hormigas siguiendo el camino BCD por unidad de tiempo será mayor que el numero de hormigas siguiendo BHD. Esto hace que se acumule más feromona en el camino más corto, y por lo tanto la probabilidad de seguir este camino aumenta rápidamente. Finalmente el camino más largo será descartado (figura c).

Basándose en esta idea, se desarrolló una técnica útil para resolver problemas de optimización combinatoria estocástica. Sus principales características son:

- retroalimentación positiva: permite el rápido descubrimiento de buenas soluciones.
- cálculo distribuido: evita la convergencia prematura.
- uso de una heurística 'greedy' constructiva: ayuda a encontrar soluciones aceptables en las primeras etapas del proceso de búsqueda.

Los 'agentes' a utilizar difieren de las hormigas reales en que:
- tendrán memoria
- no serán completamente ciegos (tendrán cierta 'inteligencia')
- vivirán en un ambiente donde el tiempo es discreto

4.2.1.2 Aplicación al Problema del Viajante (Travelling Salesman Problem)

El problema del viajante (conocido como TSP) que consiste en encontrar un ciclo cerrado de largo mínimo que visite cada ciudad una sola vez.
Consideremos el problema definiendo un grafo G = (N, A) donde N es el conjunto de ciudades a visitar y A los caminos entre pares de ciudades.
Cada arco tiene asociado un valor real dij que representa la distancia entre las ciudades ni y nj.
El objetivo es encontrar una permutación de los nodos que minimice la distancia total recorrida:

$$\sum_{i=1}^{|N|} d(n_{\pi(i)}, n_{\pi(i+1)}) \quad \text{donde } \pi(|N|+1) \equiv \pi(1)$$

Sean $b_i(t)$ (i=1,....,n) el número de hormigas en la ciudad i en el instante t y sea $m= \sum b_i(t)$ el número total de hormigas. Llamamos $camino_{ij}$ el camino más corto entre las ciudades i y j .
Cada hormiga es un agente simple con las siguientes características:
- cuando se desplaza de una ciudad i a una ciudad j deposita una sustancia llamada huella en la arista (i, j) ;
- elige la ciudad a la cual dirigirse con una probabilidad que es función de la distancia a la misma y de la cantidad de huella presente en la arista que la conecta.
- para forzar que las hormigas tomen caminos legales las transiciones a ciudades ya visitadas son inhibidas hasta que un ciclo se haya completado (lista tabú)

Sea $\tau_{ij}(t+1)$ la intensidad del rastro en el $camino_{ij}$ en el instante t+1 , dado por la fórmula

$$\tau_{ij}(t+1) = \rho . \tau_{ij}(t) + \Delta \tau_{ij}(t, t+1) \qquad (1)$$

donde ρ es un coeficiente tal que $(1-\rho)$ representa la evaporación de la huella.

$$\Delta\tau_{ij}(t, t+1) = \sum \Delta\tau^{k}_{ij}(t, t+1)$$

donde $\Delta\tau^{k}_{ij}(t, t+1)$ es la cantidad por unidad de largo de sustancia de rastro (feromona en las hormigas reales) depositada en el $camino_{ij}$ por la k-ésima hormiga entre los instantes t y t+1.

La intensidad de la huella en el instante 0, $\tau_{ij}(0)$ puede setearse a valores arbitrarios (muy pequeños en cada $camino_{ij}$).

Llamamos *Visibilidad* a la cantidad $\eta_{ij} = 1/d_{ij}$, y definimos la probabilidad de transición desde una ciudad i a otra ciudad j como

$$p_{ij}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum [\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}} & \text{si j está permitido} \\ 0 & \text{en otro caso} \end{cases} \qquad (2)$$

donde α y β son parámetros que permiten un control del usuario en la importancia relativa de la huella versus la visibilidad. Entonces la probabilidad de transición es una pugna entre la visibilidad la cual dice que ciudades cercanas son elegidas con alta probabilidad e intensidad de la huella que nos indica que si en el camino$_{ij}$ hay mucho tránsito entonces es altamente deseable.

Para satisfacer la restricción de que una hormiga visita n ciudades distintas en su recorrido, asociamos a cada hormiga una estructura de datos llamada lista tabú que memoriza las ciudades ya visitadas al instante t y evita que la hormiga las visite de nuevo antes de que un ciclo haya sido completado. Luego de completarse un ciclo la lista tabú es vaciada y liberada nuevamente para que la hormiga elija su camino.

4.2.1.3 Variantes de Ant Systems

Existen distintas instanciaciones del algoritmo de hormigas que son originadas por las diferentes alternativas de cómo calcular $\Delta\tau^{k}_{ij}(t, t+1)$ y de cuándo actualizar $\tau_{ij}(t)$. Estos son los algoritmos ANT-quantity, ANT-density y ANT-cycle.

Algoritmos ANT-quantity y ANT-density

En el modelo de ANT-quantity una cantidad constante de feromona Q_1 es dejada sobre el camino$_{ij}$ cada vez que una hormiga se desplaza de i a j .
En el modelo de ANT-density una hormiga que se desplaza desde i hasta j deposita Q_2 unidades de feromona por cada unidad de largo.

Entonces en el modelo de ANT-quantity

$$\Delta\tau^{k}_{ij}(t, t+1) = \begin{cases} \dfrac{Q_1}{d_{ij}} & \text{si la k-ésima hormiga se desplaza desde i hasta j entre los instantes t y t+1} \\ 0 & \text{en otro caso} \end{cases}$$

En el modelo de ANT-density tenemos

$$\Delta\tau^{k}_{ij}(t, t+1) = \begin{cases} Q_2 & \text{si la k-ésima hormiga se desplaza desde i hasta j entre los instantes t y t+1} \\ 0 & \text{en otro caso} \end{cases}$$

De estas definiciones se desprende que un aumento en la intensidad de la huella en la arista (i, ,j) cuando una hormiga se mueve desde i hasta j es independiente de d_{ij} en el modelo Ant-Density mientras que es inversamente proporcional a d_{ij} el modelo Ant-quantity (aristas más cortas se hacen más preferidas por las hormigas en el modelo de Ant-quantity reforzando el factor de visibilidad en la ecuación (2)).

Los algoritmos de Ant-density y de Ant-quantity son entonces:

1. Inicializar:
   t := 0
   Inicializar τ_{ij} (t) en cada camino$_{ij}$.
   Colocar b_i (t) hormigas en cada nodo i.
   $\Delta\tau_{ij}$ (t, t+1) := 0 para cada i, j.
2. Repetir hasta que la lista tabú esté llena {este paso se repetirá n veces}
   2.1. para i := 1 hasta n hacer {para cada ciudad}
   para k := 1 hasta b_i (t) hacer {para cada hormiga en la ciudad i en el instante t}
   Elegir la ciudad hacia la cual dirigirse, con probabilidad p_{ij} dada por la ecuación (2) y mover la k-ésima hormiga a la ubicación elegida.
   Insertar la ciudad elegida en la lista tabú de la hormiga k.
   $\Delta\tau_{ij}(t, t+1) := \Delta\tau_{ij}(t, t+1) + \Delta\tau^{k}_{i\,j}(t, t+1)$ donde $\Delta\tau^{k}(t, t+1)$ está definido en (3) o (4).
   2.2. calcular τ_{ij} (t+1) y p_{ij} (t+1) de acuerdo a ecuaciones (1) y (2).
3. Memorizar el camino más corto encontrado hasta ahora y vaciar todas las listas tabú.
4. Si no (Terminar) entonces
   t := t + 1
   $\Delta\tau_{ij}$ (t, t+1) := 0 para cada i, j.
   Ir a 2.
   si no
   Imprimir el camino más corto y parar. { Terminar indica una prueba en la cantidad de ciclos}

El algoritmo trabaja así:
En el instante cero tiene lugar una fase de inicialización durante la cual las hormigas son colocadas en diferentes ciudades y se le asignan valores iniciales de intensidad de huella en las aristas.
El primer elemento de la lista tabú de cada hormiga es seteado en la ciudad de comienzo.

De ahí en más cada hormiga se desplaza de una ciudad i a otra ciudad j eligiendo la ciudad destino con una probabilidad que es dada por la función p_{ij}.
Cada vez que una hormiga realiza un desplazamiento la huella que deja en la arista (i,j) se suma a la huella dejada en la misma arista en el pasado.
Cuando todas las hormigas se han desplazado las probabilidades de transición se calculan usando los nuevos valores de huella de acuerdo a las fórmulas (1) y (2).
Luego de n-1 movidas la lista tabú de cada hormiga será llenada: el camino más corto encontrado por las m hormigas se memoriza y todas las listas tabú son vaciadas.
Este proceso es repetido hasta que el contador de vueltas alcanza el máximo definido por el usuario o todas las hormigas realizan el mismo camino.

Complejidad:

La complejidad de estos algoritmos es O (NC . n^3) donde NC es el número de ciclos (podría ser una función de n).

El algoritmo Ant-cycle:

En este caso se introduce una diferencia fundamental respecto de los dos sistemas anteriores. Aquí $\Delta\tau^k_{ij}$ no se calcula en cada paso sino después de una vuelta completa (n pasos). El valor de $\Delta\tau^k_{ij}$ (t, t+1) está dado por:

$$\Delta \tau^k_{ij}(t, t+n) = \begin{cases} \dfrac{Q_3}{L^k} & \text{si la k-ésima hormiga usa la arista (i, ,j) en su ciclo} \\ 0 & \text{en otro caso} \end{cases}$$

donde Q_3 es una constante y L^k es el largo del ciclo de la k-ésima hormiga.
esto corresponde a una adaptación del método de Ant-quantity, donde las huellas son actualizadas al final de todo el ciclo y en vez de después de cada movida simple. Se espera que este algoritmo tenga una mejor performance que para los anteriores debido a que aquí se utiliza información global sobre el valor del resultado (el largo del ciclo).

El valor de la huella también es actualizado cada n pasos de acuerdo a una fórmula muy similar a (1)

$$\tau_{ij}(t+n) = \rho_1 . \tau_{ij}(t) + \Delta\tau_{ij}(t, t+n) \qquad (1')$$

donde $\Delta\tau_{ij}(t, t+n) = \sum_{k=1}^{m} \Delta\tau^k_{ij}(t, t+n)$

y ρ_1 es diferente de ρ debido a que la ecuación no es actualizada en cada paso sino sólo después de un ciclo (n pasos).

Algoritmo:

1. Inicializar:
   t := 0
   Inicializar τ_{ij} (t) en cada camino$_{ij}$.
   Colocar b_i (t) hormigas en cada nodo i.
   $\Delta\tau_{ij}$ (t, t+n) := 0 para cada i, j.
2. Repetir hasta que la lista tabú esté llena {este paso se repetirá n veces}
   2.1. para i := 1 hasta n hacer {para cada ciudad}
   para k := 1 hasta b_i (t) hacer {para cada hormiga en la ciudad i en el instante t}
   Elegir la ciudad hacia la cual dirigirse, con probabilidad p_{ij} dada por la ecuación (2) y mover la k-ésima hormiga a la ubicación elegida.
   Insertar la ciudad elegida en la lista tabú de la hormiga k.
   2.2. calcular $\Delta\tau^k_{ij}$ (t, t+n) como está definido en (5).
   2.3. calcular $\Delta\tau_{ij}(t, t+n) = \sum_{k=1}^{m} \Delta\tau^k_{ij}(t, t+n)$
   2.4. calcular τ_{ij} (t+n) y p_{ij} (t+n) de acuerdo a ecuaciones (1') y (2).
3. Memorizar el camino más corto encontrado hasta ahora y vaciar todas las listas tabú.
4. Si no (Terminar) entonces
   t := t + n
   $\Delta\tau_{ij}$ (t, t+n) := 0 para cada i, j.
   Ir a 2.
   si no
   Imprimir el camino más corto y parar. { Terminar indica una prueba en la cantidad de ciclos}

Complejidad:

La complejidad de este algoritmo es O (NC . n^3) donde NC es el número de ciclos (podría ser una función de n).

4.2.2 Algoritmos genéticos

Descripción de la técnica:

Un problema de optimización se traduce en el problema de encontrar el individuo más apto (a veces llamado cromosoma) dentro de una población. La aptitud se mide por medio de una función de aptitud (fitness) la cual se relaciona con la función objetivo del problema a resolver. Los individuos (cromosomas) son equivalentes a soluciones y una población es un conjunto de N individuos.

Cada individuo consiste en una secuencia (usualmente un string) de elementos atómicos llamados genes. Cada gen puede tomar valores (alelos) en un conjunto

predefinido. Es usual que cada gen tome valores en el conjunto {0,1} o en el conjunto de los dígitos.

El algoritmo genético actúa sobre la población de cromosomas modificando sus componentes. Las modificaciones ocurren de acuerdo a las reglas de la genética a través de la aplicación de operadores genéticos.

El operador Reproducción obtiene un nuevo individuo tomando en cuenta el fitness de cada individuo en la población actual, es decir de la población actual elige de acuerdo al fitness de los individuos cuáles se van a reproducir en la población siguiente (Newpop).

El operador Crossover toma como entrada la nueva población NewPop y devuelve otra población CrossPop. Dicho operador extrae aleatoriamente dos individuos (padres), elige con probabilidad de distribución uniforme un punto de cruzamiento en los cromosomas que representan a los dos padres y entonces intercambia los valores a la derecha de este punto recombinándolos y generando dos hijos. El operador CrossOver es aplicado con una probabilidad p_c independiente de los individuos específicos sobre los cuales es aplicado. Sirve para crear nuevos individuos que preserven las mejores características de sus padres.

El operador Mutación toma como entrada la población CrossPop y devuelve una nueva población MutPop en la cual algunos individuos han sufrido mutaciones. Cada individuo puede ser seleccionado con probabilidad p_m y aquellos que lo son sufren alguna mutación. Por ejemplo, en el caso que los individuos estén codificados como strings de ceros y unos la mutación cambia un uno por cero o viceversa. El operador mutación introduce variaciones de base en la población garantizando la posibilidad de explorar todo el espacio de búsqueda independientemente de la población inicial específica.

Para resolver un problema de optimización usando algoritmos genéticos debemos elegir la estructura de los cromosomas para codificar en ellos las soluciones. Se debe determinar la función de fitness como función de los valores de los genes, la cual estará directamente relacionada con la función objetivo del problema a resolver. Para hallar la solución se aplican sucesivamente los tres operadores estudiados a cada población resultante hasta que finalmente la población final estará solo compuesta por un tipo de individuos, el más apto, que constituirá la solución al problema.

Aplicación al problema:

Esta técnica se basa en la manipulación de un conjunto de soluciones al problema. Dada la complejidad de la estructura de estas, parece complejo poder generar varias soluciones de partida para el algoritmo, así como poder aplicar efectivamente el operador de ‘CrossOver’ entre dos soluciones. Sin embargo se ha aplicado con éxito a problemas de este tipo.

4.2.3 Simulated Annealing

Descripción de la técnica:

Es básicamente un método de búsqueda local. Partiendo de una solución factible inicial 'i' se genera mediante algún procedimiento una solución vecina $j \in V(i)$ donde V(i) es el conjunto de soluciones vecinas a 'i'. En esta heurística, la solución 'j' se aceptará como la nueva solución factible con cierta probabilidad aún cuando la misma sea peor que la solución de partida 'i'. Si consideramos una función objetivo a minimizar 'f', sea $\Delta = f(j) - f(i)$. La probabilidad de aceptar a 'j' como la nueva solución está dada por:

$$\text{Prob (aceptar j)} = \begin{cases} 1 & \text{si } \Delta < 0 \\ e^{-\Delta/t} & \text{si } \Delta \geq 0 \end{cases}$$

Donde 't' es un parámetro de control del algoritmo, a menudo llamado 'temperatura'.

Este proceso se repite para un valor dado de 't' hasta que se llegue a una condición de *equilibrio* que dependerá del problema. En su forma más simple esta condición se reduce a un numero fijo de iteraciones. Luego de llegar al equilibrio la temperatura 't' se decrementa de acuerdo a alguna regla y el proceso vuelve a comenzar.

El algoritmo termina al llegar a un valor mínimo de 't' establecido, en el que prácticamente 'j' se acepta sólo si $\Delta < 0$; en este punto el algoritmo actúa como un algoritmo simple de búsqueda local.

El conjunto de parámetros que incluye temperatura inicial, final, regla para decrementar la temperatura y condición de equilibrio se conoce como 'estrategia de enfriamiento', y ésta es fundamental para que el algoritmo sea exitoso.

Aplicación al problema:

Podría aplicarse de forma bastante sencilla, pero estudios ya realizados [6][12] muestran que es una técnica que requiere tiempos computacionales bastante largos y es inferior a otras heurísticas en el problema de 'Timetable'.

4.2.4 Sampling and Clustering

Descripción de la técnica:

Se basa en la heurística llamada 'Multistart' (MS). Esta consta de dos fases básicas: una 'global' y otra 'local'. La fase global genera al azar un número K de soluciones de partida en el espacio de soluciones S. La fase local consiste en aplicar un método de búsqueda local a cada punto (solución) de partida, lo cual genera un conjunto de soluciones localmente óptimas que se suma a las ya existentes. El proceso se repite hasta que una fase de 'testeo' lo detiene, cuando el conjunto de óptimos locales no cambia en las últimas búsquedas locales, o bien luego de un número prefijado de iteraciones.
En esta heurística ocurre que las soluciones de partida generadas al azar a menudo llevan al mismo óptimo local. Sampling and Clustering intenta evitar este problema

identificando ‘clusters’ (grupos) de soluciones iniciales que llevarán al mismo óptimo local, evitando así la repetición innecesaria de la fase local.

Para la aplicación de SC deben definirse:

I = conjunto de puntos iniciales a partir de los que se hará la búsqueda local.
O = conjunto de soluciones óptimas localmente.
C = conjunto de soluciones a partir de las cuales se llega por medio de la búsqueda local a algún elemento de O previamente encontrado.

Definimos $O \cup C$ = conjunto de puntos ‘semilla’.

Los conjuntos I, O y C se inicializan en $\varnothing$.

Fase Global:
- generar al azar K nuevos puntos de S y agregarlos a I
- eliminar de I un subconjunto de puntos con los peores valores de la función objetivo
- agrupar (cluster) tantos puntos como sea posible con puntos semilla, de acuerdo a una determinada regla (conocida como ‘clustering rule’).

Fase Local:
Si quedan puntos de I sin agrupar comienza esta fase, utilizando los mejores candidatos de estos puntos para efectuar búsquedas locales, hasta llegar a un óptimo local. Si se llega a un nuevo óptimo, éste se agrega a O, de lo contrario el punto de partida se agrega a C.

Fase de Testeo:
Detiene el algoritmo cuando no se han encontrado nuevos óptimos locales, es decir, O no ha cambiado.

Estos algoritmos dependen fuertemente de asumir que las soluciones pueden ser aleatoriamente generadas uniformemente en S. para muchos COP es fácil generar aleatoriamente soluciones uniformemente , pero no es fácil generar aleatoriamente soluciones factibles uniformemente .en estos casos se debe decidir si el método debe perder tiempo chequeando factibilidad y penalizando a las soluciones no factibles o si es preferible relajar la asunción de uniformidad.
El método SC se adecua más a problemas para los cuales el costo de calcular un óptimo local de una solución dada inicial es en promedio superior que el costo de agrupar una solución dada.

Aplicación al problema:

Esta técnica se basa fuertemente en la posibilidad de generar soluciones al azar en forma uniforme en el espacio de soluciones [6]. En nuestro problema se hace difícil generar estas soluciones de modo que sean factibles o al menos un buen punto de partida para llegar a un óptimo local. Una posibilidad es relajar la exigencia de uniformidad y generar las soluciones basándonos en una cierta preasignación de algunas de las clases.

4.2.5 Tabu Search

Descripción de la técnica:

Las técnicas de búsqueda local son una familia de técnicas de propósito general para la solución de problemas de optimización. Están basadas en la noción de vecindad. Consideremos un problema de optimización y sea S su espacio de búsqueda y 'f' su función objetivo a minimizar. Una función 'N' la cual depende de la estructura del problema particular asigna a cada solución factible $s \in S$ su vecindad $N(s) \subseteq S$. Cada solución $s' \in N(s)$ se llama vecino de s.

Una técnica de búsqueda local comenzando en una solución inicial s_0 entra en un loop y navega a través del espacio de búsqueda pasando iterativamente de una solución a alguna de sus vecinas. Se llama 'movida' a la modificación que transforma una solución en una de sus vecinas.

Partiendo de una solución inicial s_0 el algoritmo explora un subconjunto V del vecindario N(s) de la solución actual s. El elemento de V que hace mínimo el valor de la función objetivo pasa a ser la nueva solución sin importar que su valor sea peor o mejor que el valor en s. Para prevenir ciclos se maneja una lista de movimientos que no pueden realizarse, llamada lista 'tabú'. Esta lista contiene los últimos k movimientos realizados. Existe además un mecanismo que puede modificar el estado tabú de una movida: si una movida brinda una gran mejora al valor de la función objetivo, entonces su estado tabú es borrado y la solución resultante pasa a ser la nueva solución actual. Mas precisamente se define una 'función de aspiración' A que, para cada valor v de la función objetivo devuelve otro valor v' que representa el valor al cual el algoritmo aspira a llegar desde v. Dados una solución actual s, la función objetivo f, y una solución vecina s' obtenida a través de una movida m, si f(s') <= A(f(s)) entonces s' es aceptada aún cuando m esté en la lista tabú.

El procedimiento finaliza luego de un numero prefijado de iteraciones sin que mejore el valor de la función objetivo , o bien cuando la función objetivo alcanza una cota mínima.

Los parámetros que controlan el algoritmo son: el largo de lista tabú, la función de aspiración, la cardinalidad del conjunto V y la máxima cantidad de iteraciones sin mejorar el valor de la función objetivo.

Aplicación al problema:

Puede aplicarse fácilmente a nuestro problema ya que se basa en la generación de una única solución inicial (al igual que SA), la cual se usa como punto de partida para una búsqueda local (neighbourhood search). Además resulta natural navegar en el espacio de soluciones realizando pequeñas modificaciones en la solución actual. Según estudios realizados [12], esta heurística permite obtener mejores resultados que SA en este tipo de problemas.

4.2.6 Neural Nets

Descripción de la técnica:

Esta técnica se basa en imitar el comportamiento y forma de comunicación de las neuronas reales. Una red esta formada por un conjunto de elementos simples (neuronas) interconectados. Cada uno tiene como entradas las señales provenientes de otras neuronas que estén conectadas a él , ponderadas por un coeficiente w_{ij} . Como salida emitirán una señal de acuerdo a las suma de todas las señales entrantes, si la intensidad o valor de ésta sobrepasa una cierta cota θ (llamada threshold).
En otros modelos la salida es una función más o menos compleja de todas las entradas.

La red evoluciona a través de iteraciones sucesivas según una regla de transición que permite ir modificando el estado de cada neurona, hasta llegar a un punto en que el estado de toda la red no cambia; en ese momento se habrá llegado a un estado estable, y por lo tanto a una solución. Existe otra variante en la cual la evolución de la red no está dada por un estado propio de cada neurona sino por ña modificación de los pesos asociados a cada conexión entre las mismas.

Aplicación al problema:

Las redes neuronales han sido aplicadas a problemas de asignación del tipo asignar recursos a localidades o al QAP. Un ejemplo de como podría aplicarse puede encontrarse en [6] . En el citado ejemplo , se trata de ubicar P servicios que deberán ser accedidos por N usuarios distribuidos en el plano(P < N) ,minimizando las distancias entre servicios y usuarios. Cada usuario tendrá acceso a un sólo servicio. La asignación puede representarse mediante una matriz W donde las filas representan a los usuarios y las columnas a los servicios. Cada elemento w_{ij} será igual a 1 si al usuario i se le asignó el servicio j. Por cada elemento w_{ij} de la matriz se define una neurona ,la cual se inicializa en un estado con valor 1/P con cierto ruido aleatorio. A través de las sucesivas iteraciones del algoritmo y la regla de actualización del estado ,que incluye la función objetivo de minimizar distancias ,la red evolucionará hasta un estados estable en el que cada neurona w_{ij} tomará un valor booleano (0 o 1) indicando la asignación resultante. Luego de acuerdo a esta asignación se calcularán las posiciones en las que deberán ubicarse los servicios en el plano.

Observación

Las técnicas de búsqueda local tienen la ventaja de permitir interactividad, es decir, puede partirse de una determinada solución inicial pre-elaborada o bien modificarse la misma a medida que se va construyendo [12].
Estas técnicas son utilizadas por SA o TS.

5 Diseño utilizando Ant Systems

En los puntos 3.4.2. y 3.4.3. se estudió la analogía existente entre el problema de asignación cuadrático (QAP) y el problema de asignación de salones y horarios a cursos. El diseño de la solución a nuestro problema se basa en la analogía mencionada y en la resolución del QAP por medio de la heurística Ant Systems.

5.1 Resolución del QAP utilizando la heurística Ant Systems

Para explicar el mecanismo de resolución del QAP por medio de Ant Systems, retomaremos el ejemplo visto en el punto 3. En ese ejemplo, se busca minimizar el costo de asignar servicios a localidades, considerando que los servicios se caracterizan por el grado de relacionamiento entre los mismos (los llamados flujos entre servicios), y las localidades tienen diferentes distancias entre sí. Por otra parte, existe un costo por asignar un servicio determinado a una localidad determinada.

Formalmente el problema se define por tres matrices nxn:

$D_{nxn} = \{d_{ih}\}$ = distancia entre la localidad i y la localidad h.
$F_{nxn} = \{f_{jk}\}$ = flujo entre el servicio j y el servicio k.
$C_{nxn} = \{c_{ij}\}$ = costo de asignar el servicio j a la localidad i.

Recordando las matrices de distancias y flujos tenemos:

$$D = \begin{vmatrix} 0 & 1 & 1 & 2 & 3 \\ 1 & 0 & 2 & 1 & 2 \\ 1 & 2 & 0 & 1 & 2 \\ 2 & 1 & 1 & 0 & 1 \\ 3 & 2 & 2 & 1 & 0 \end{vmatrix} \qquad F = \begin{vmatrix} 0 & 5 & 2 & 4 & 1 \\ 5 & 0 & 3 & 0 & 2 \\ 2 & 3 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 5 \\ 1 & 2 & 0 & 5 & 0 \end{vmatrix}$$

Veremos una forma de determinar qué tan buena es una asignación de servicios a localidades [5]. Para ello se definen dos vectores, P_d y P_f llamados potencial de distancias y potencial de flujos respectivamente.

El potencial de flujo de cada servicio se calcula en base a la matriz F, sumando todas sus columnas, y el potencial de distancias de cada localidad se calcula en base a la matriz D, sumando todas sus columnas.

Los valores de potencial de distancia indican la suma de distancias entre un nodo particular y todos los demás. Los valores del potencial de flujo indican el flujo total intercambiado entre un servicio y todos los demás.

A menores valores de d_i (potencial de distancia) de la localidad "i" más baricéntrica es ésta en el conjunto de localidades. A mayor valor de f_j (potencial de flujo) del servicio "j", más importante es este servicio en el sistema de flujos intercambiados.

$$F = \begin{vmatrix} 0 & 5 & 2 & 4 & 1 \\ 5 & 0 & 3 & 0 & 2 \\ 2 & 3 & 0 & 0 & 0 \\ \end{vmatrix} \qquad P_f = \begin{vmatrix} 12 \\ 10 \\ 5 \\ 9 \end{vmatrix}$$

$$D = \begin{vmatrix} 0 & 1 & 1 & 2 & 3 \\ 1 & 0 & 2 & 1 & 2 \\ 1 & 2 & 0 & 1 & 2 \\ 2 & 1 & 1 & 0 & 1 \\ 3 & 2 & 2 & 1 & 0 \end{vmatrix} \qquad P_d = \begin{vmatrix} 7 \\ 6 \\ 6 \\ 5 \\ 8 \end{vmatrix}$$

Determinación de la visibilidad

Una buena permutación podría lograrse entonces asignando los servicios a las localidades según la regla “mín-máx”, es decir, los servicios con mayor potencial se asignan con alta probabilidad a localidades de bajo potencial (entidades con una ubicación central). Podemos obtener una ‘matriz de acoplamiento’ de la siguiente manera:

$A = P_d \times P_f{}^t$

cuyos elementos son $a_{ij} = d_i \times f_j$. Las filas de A representan localidades y las columnas servicios.

$$A = \begin{vmatrix} 84 & 70 & 35 & 63 & 56 \\ 72 & 60 & 30 & 54 & 48 \\ 72 & 60 & 30 & 54 & 48 \\ 60 & 50 & 25 & 45 & 40 \\ 96 & 80 & 40 & 72 & 64 \end{vmatrix}$$

El algoritmo entonces recorrerá las columnas (servicios) de A en orden de potencial decreciente, y para cada uno de ellos seleccionará la localidad con menor potencial, asumimos aquí que la elección de localidad se efectúa en forma determinística para simplificar(y no probabilística como sucede en el algoritmo).
Por lo tanto la visibilidad es entonces $\eta_{ih} = 1/a_{ih}$.

Cada fila de A se considera una sola vez, de modo de no repetir localidades en la asignación.

$$A = \begin{array}{c} \begin{array}{ccccc} 1º & 2º & 5º & 3º & 4º \end{array} \\ \begin{vmatrix} 84 & 70 & 35 & 63 & \underline{56} \\ 72 & 60 & 30 & \underline{54} & 48 \\ 72 & \underline{60} & 30 & 54 & 48 \\ \underline{60} & 50 & 25 & 45 & 40 \\ 96 & 80 & \underline{40} & 72 & 64 \end{vmatrix} \end{array}$$

(servicios considerados en orden de potencial)

En este ejemplo, se comienza por el servicio de mayor potencial, que corresponde a la primera columna de la matriz, y se le asigna la localidad con menor potencial de distancia. En la primera columna de la matriz, el menor elemento es 60, correspondiente a la fila 4 de la matriz, o sea la localidad 4. Análogamente se asigna la localidad 3 al servicio 2. Al asignar el servicio 4 (el 3º en importancia) no se le puede asignar la localidad 4 (que sería la de menor valor) pues ya fue asignada al servicio 1. Por lo tanto se le asigna la localidad 2 (de valor 54).

El espacio de soluciones a recorrer puede representarse mediante una lista de servicios en un cierto orden fijo, y una lista de localidades. La visibilidad η_{ij} estaría representada por la 'bondad' o conveniencia de asignar el servicio 'j' a la localidad 'i'.

A fin de simplificar el razonamiento, asumimos que la hormiga se guía por una regla codiciosa determinística, y no probabilística. Las localidades ya utilizadas por la hormiga se almacenan en su lista tabú, de modo que no puedan repetirse localidades en la asignación. **Debemos notar que en la consideración de cada servicio podremos determinar las visibilidades de ese servicio a cada una de las localidades candidatas a formar un par en la asignación.**

De este modo la secuencia de parejas de (servicio, localidad) representa una asignación, o sea el camino elegido por la hormiga.

Ejemplo:

Si la lista de servicios en orden de 'potencial de flujo' es: A,B,C,D,E y la secuencia de localidades elegidas es :L1, L2, L4, L5, L3

La asignación correspondiente es:

A - L1
B - L2
C - L4
D - L5
E - L3

5.2 Resolución del problema de asignación utilizando Ant Systems

5.2.1 Analogía con el QAP

Nuestro problema puede identificarse fácilmente con el problema de asignación cuadrático (QAP), ya que se trata de asignar salones y horarios a clases. La diferencia principal consiste en que nuestro problema consta de tres dimensiones básicas (salones, horarios y clases) en lugar de dos (servicios y localidades) y la función objetivo se calcula de forma distinta, lo cual determinará un cálculo distinto de la visibilidad.

Las tres dimensiones pueden reducirse a dos agrupando dos de ellas. A continuación se analizan las distintas posibilidades que surgen.

5.2.2 Representación del Problema de Asignación de Salones y Horarios a Cursos

5.2.2.1 Determinación de la forma de representar una asignación

Al agrupar dos de las entidades y considerarlas como una nueva, debemos determinar entre las dos entidades resultantes, cual de ellas servirá de “maestra” para la realización de la asignación (correspondiendo a los ‘servicios’ en una analogía con el QAP) y cual de ellas será sobre la cual elegiremos candidatos (correspondiente a las ‘localidades’ en una analogía con el QAP), que de ahora en adelante llamaremos entidad “detalle”.

Existen varias combinaciones posibles:

1) Si consideramos el espacio agrupando salones y horarios (s,h), el problema de asignación se reduciría a la asignación de clases a parejas de (salón, horario).
2) Si consideramos el espacio agrupando clases y horarios (c,h), el problema se reduciría a la asignación de salones a parejas de (clase, horario). Esta altenativa no parece ser interesante ya que para la mayoría de las parejas de clase y horario no tiene sentido pensar en asignarle un salón ya que una clase puede ser asignada a sólo un salón y a unos pocos horarios (consecutivos). Tampoco parece natural partir de los salones y asignarles parejas de horario y clase.
3) Un razonamiento análogo nos permite descartar la consideración de la agrupación de entidades clase y salón (c,s) como una de las entidades y los y horarios como la otra.

La alternativa 1) sería la que tendría sentido, ya que para cada pareja de salón y horario es muy probable que se le pueda hacer corresponder una clase, y sí es evidente que para cada clase nos interesa asignarle una pareja de salón y horario.

Considerando esta agrupación tenemos dos maneras de recorrer el espacio de soluciones para construir las componentes de la asignación:

a. considerar las clases en orden de importancia (equivalentes a los 'servicios') y las parejas (salón, horario) como candidatas o elementos de la entidad "detalle" (equivalentes a las 'localidades').

b. considerar parejas de (salón, horario) en un orden fijo según algún criterio y las clases como entidad detalle. Esta opción es la inversa de la anterior.

A continuación se estudian estas dos opciones.

En la representación a., dada una clase, se debe determinar la pareja (salón, horario) a la cual asignarla. La entidad detalle constará de tantos nodos como combinaciones de salones y horarios existan. En la representación opuesta, la lista ordenada a recorrer (entidad maestra) será de parejas (s, h) y la entidad detalle constará de clases.

Estimemos la cantidad de elementos de cada entidad para el problema que estamos tratando:

- horarios: si consideramos un rango horario diario de 8:00 a 23:00 y 5 días de clase a la semana tendremos que la cantidad de medias horas es:
  (23 - 8) * 5 * 2 = 150.
- salones: 20 aproximadamente.
- clases: 600 aproximadamente.

De modo que si la entidad detalle está representada por parejas (s, h) la cantidad de elementos de la misma sería 20 * 150 = 3000; si en cambio la entidad detalle está formada por las clases, la cantidad de elementos sería aproximadamente 600.

Esto implica que en el caso a. se tiene una cantidad de elementos mucho mayor entre los cuales elegir la asignación que en el caso b., lo cual implica que en cada paso sea mucho mayor el espacio a examinar. **También es más probable que existan varias opciones con probabilidades similares,** y esto puede dificultar la elección de la que globalmente resulte la mejor.

En la opción a., al elegir una pareja (s, h) para una clase c deben asignarse a la misma clase las parejas que consten del mismo salón y horarios subsiguientes hasta abarcar la duración total de la clase. Esto se traduce en que quedará determinado en la secuencia de asignación un tramo compuesto por ternas donde a la clase c se le hacen corresponder parejas del mismo salón y horarios consecutivos a partir del inicial, según la duración de la clase. Este hecho en la opción b. hace que una vez que se determinó una clase para la pareja (s,h) que se está considerando, debo extraer de la lista de (s,h) las parejas que corresponden a los horarios que abarcará la clase.

En la opción b. los salones y horas se recorrerán en cierto orden. Qué ocurriría si la cantidad de clases fuera menor que la de parejas (s,h) ? Todas las clases se asignarían en las primeras parejas consideradas, quedando inexplorado parte del espacio de parejas (s,h). De este modo las clases se asignarían en forma 'codiciosa', sin considerar zonas del espacio (s,h) donde podría haberse obtenido una mejor

asignación. Este hecho no es subsanado por el no determinismo del algoritmo, ya que en un paso puede hacerse una asignación u otra con cierta probabilidad, pero **siempre** se hace una asignación. Por otro lado, esta estrategia permitiría realizar una asignación compacta, sin 'puentes'. Una manera de evitar asignar clases en forma temprana es, si no existe una clase suficientemente atractiva para la pareja (s,h), saltear esta pareja e intentar asignarla más tarde, en otra recorrida del espacio. Otra opción es recorrer las parejas (s,h) según un orden de importancia o demanda; esto a su vez podría generar puentes al no recorrer los horarios en orden creciente.

La estrategia utilizada en a. asegura que al considerar una clase, elegiré teniendo en cuenta la totalidad del espacio (s,h) disponible, dando la mejor asignación de (s,h) según la clase considerada (si obviamos el no determinismo de la heurística). En este caso, no es tan probable que se logre una asignación compacta. Para subsanar esto, podría considerarse la información histórica de las asignaciones realizadas hasta el momento, de forma de evitar 'puentes' al elegir (s,h). La ventaja de esta estrategia es que recorrería el espacio de parejas (s,h) en forma total y permitiría, si consideramos las clases según un orden de importancia, dar buenas asignaciones a las clases que más lo requieran, de forma natural.

El espacio de soluciones en ambos casos debe representarse con una estructura que permita realizar recorridas en la entidad maestra y otra que permita elegir candidatas en la entidad detalle .

En principio si consideráramos la opción a. se tendría una lista de las clases ordenadas según importancia, que contendría la información relevante a cada clase y luego una lista con aproximadamente 3000 elementos.

En el otro caso se tendrá una lista de salones y horarios ordenada por algún criterio con 3000 elementos y la lista de clases (entidad detalle) con 600 elementos.

El espacio de almacenamiento que debería destinarse para registrar los valores de las visibilidades no afectará el tamaño efectivo de las estructuras de datos consideradas si las preferencias que intervienen el cálculo de la visibilidad son calculadas en el momento de sortear la clase entre las candidatas.

El espacio a almacenar entonces no sería más grande en la opción b.

Pero en el caso a. se tiene una cantidad de elementos mucho mayor entre los cuales elegir (pareja de salón y horario) para formar una terna en la asignación que en el caso b., lo cual implica que en cada paso sea mucho mayor el espacio a examinar. Lo más determinante sería que dada una clase existirían varias opciones con probabilidades similares, y esto puede dificultar la elección de la que globalmente resulte la mejor.

En el caso a. el paso de elección será mucho más corto mientras que en el caso b. sólo se recorrerá menos veces la entidad maestra.

Tomar como entidad maestra las parejas de salón y horario puede favorecer el encuentro de soluciones compactas (con pocos puentes). Tiene el defecto de que se debe controlar la codicia del método ya que para cada pareja de salón y horario debe elegir entre las clases la mejor, en caso de que haya alguna interesante, o elegir la clase vacía en caso de no encontrar opciones que satisfagan ciertas preferencias.

La gran ventaja de considerar una entidad detalle con pocas instancias será tener que efectuar recorridos más cortos y menos opciones en cada paso del algoritmo logrando de esta manera mayor rapidez. Dada una pareja (s, h), no son muchas las clases que sean buenas candidatas; en la otra opción, dada una clase probablemente existan más parejas (s, h) atractivas y menos probabilidad quizás de acertar con las mejores elecciones ya que hay más opciones entre las que elegir.

Optamos entonces por la opción b., que consiste en recorrer las parejas de salón y horario y elegir las clases.

5.2.2.2 Elección del próximo nodo

Como ya se ha expuesto en la sección 4., la determinación del próximo nodo que visitará la hormiga se hace en función de una regla probabilística que involucra dos elementos: la 'visibilidad' y el 'rastro' dejado en recorridos anteriores. A continuación veremos cómo calcular cada uno de estos elementos.

5.2.2.2.1 Visibilidad

La visibilidad se calculará en base a las preferencias (descritas en el punto 3.) que son las que conforman parte de la función objetivo.

Podemos clasificar las preferencias en dos grupos:

- **locales:** son preferencias que no dependen de los pasos anteriores o futuros de la 'hormiga', es decir, no dependen de cómo se ha construido la solución hasta el momento. Estas podrían ser:
    - capacidad del salón
    - horarios de preferencia
    - otros requerimientos
    - desperdicio de espacio en el salón

- **globales:** son las que relacionan distintas ternas de la asignación, y por lo tanto dependen de cómo se ha venido construyendo la solución. Estas podrían ser:
    - no superposición entre materias
    - adyacencia
    - mismo salón
    - espaciamiento en días

El cálculo de las visibilidades podría hacerse en forma previa al comienzo del recorrido de las hormigas, y tendríamos una visibilidad 'estática', independiente de cómo se construya la solución. O podría calcularse en forma 'dinámica', cada vez que un nuevo nodo deba ser elegido. Las preferencias locales son las más aptas para calcular en forma 'estática', mientras que las globales deben calcularse en base a la solución que se está construyendo, ya que relacionan ternas entre sí.

El cálculo de la visibilidad en forma estática será más económico al no tener que hacerse todos los cálculos en cada paso del algoritmo, sino una sola vez en forma previa. Por otro lado, esta opción puede conducirnos a un algoritmo menos inteligente,

ya que aún las preferencias locales podrían llegar a modificarse según se construye la solución, es decir, podrían ser ‘dinámicas’. Por ejemplo la preferencia de ‘capacidad del salón’ puede tener distinta fuerza o relevancia dependiendo de los salones que se hallen disponibles según lo asignado hasta el momento.

El cálculo en forma dinámica es más costoso, pero permite tomar decisiones más inteligentes.

Suponiendo que se controlen las restricciones a la solución a medida que ésta se construye, en el momento de determinar el siguiente nodo, existirán algunos de ellos que no se seleccionarán ya que no conducirán a una solución factible. Por lo tanto, si tuviéramos la visibilidad calculada en forma estática, habrían cálculos correspondientes a los nodos descartados que no se utilizarían.

5.2.2.2.2 Rastros

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. De esta forma se hace en las variantes Ant Quantity y Ant Density.
- el rastro de todas las aristas recorridas por la hormiga se actualiza luego que la misma terminó su recorrido. Esta es la estrategia usada en Ant Cycle.

En su forma más simple, el rastro sólo indica que una arista fue recorrida, no importando qué tan bueno haya sido elegir esa arista o qué tan buena resulte ser la solución construida. Es decir, en este sentido todos los caminos tomados tienen la misma 'bondad'.

Una actualización del rastro que resultaría más beneficiosa, podría tener en cuenta qué tan bueno es haber elegido un cierto nodo en un paso del recorrido, es decir, evaluaría el recorrido en forma *local*.

La estrategia utilizada en Ant Cycle evalúa el recorrido en su totalidad, en forma *global*. En el caso del TSP, se utiliza el largo total del recorrido como indicador de la 'bondad' de la solución.

En nuestro problema, dada la complejidad de la función objetivo, algunos de cuyos componentes evalúan relaciones entre las distintas ternas que componen la asignación, es necesario evaluar el recorrido de la hormiga en forma global. No se puede obtener una buena medida de cuán buena es la solución evaluando en forma separada cada terna que compone la asignación. Por lo tanto la variante de Ant Systems a utilizar será **Ant Cycle**, y la cantidad de rastro dejada por una hormiga dependerá del valor que tome la función objetivo para esa solución. Por lo tanto los rastros se actualizarán una vez que la hormiga finalizó su recorrido.

Alternativas posibles

Una alternativa posible es calcular la visibilidad en forma anticipada en base a las preferencias locales (visibilidad 'estática') y obtener el rastro en base a la función objetivo, la cual engloba todas las preferencias. Esta opción tiene la ventaja de ser económica pero no es la que aporta más inteligencia al algoritmo.

Otra posibilidad es calcular la visibilidad en el momento de elegir el próximo nodo. Esto permite una mejor evaluación de las posibilidades pero resulta en una mayor carga para cada paso de la heurística.

5.2.2.2.3 Restricciones

Consideraremos en este punto cómo y cuándo controlar la factibilidad de las soluciones construidas.

Las restricciones, que indican si una solución es factible o no, pueden dividirse en:

- Restricciones que se cumplen automáticamente dada la representación elegida.
- restricciones locales: no dependen del resto de las ternas que componen la solución
- restricciones globales: dependen de la composición de la solución, es decir, relacionan ternas entre sí.

Las restricciones locales pueden controlarse en el momento de elegir una terna (salón, horario, clase), es decir, cuando la hormiga en uno de sus pasos elija una determinada clase para una pareja (s, h). Si la única elección posible que tiene la hormiga viola una restricción, entonces la solución que se está construyendo deja de ser factible. En ese caso, no tiene sentido permitir que la hormiga continúe su recorrido; la solución debe ser descartada.

Las restricciones globales en cambio, no pueden controlarse hasta que la hormiga finalizó su recorrido, completando la asignación. Es decir, no puede saberse si la solución es factible hasta que se construyó totalmente.

Es deseable entonces que la mayoría de las restricciones sean ‘locales’, para lograr una mayor eficiencia.

5.2.2.2.4 Mejoras al algoritmo básico

Asignación en bloque

En el problema que estamos tratando es importante lograr ciertas características en la asignación que podríamos llamar ‘simetrías’. Por ejemplo, es conveniente que las clases que se dictan en la semana de un mismo grupo de un curso tengan todas el mismo horario de comienzo, y que se dicten en el mismo salón. Si tenemos que una clase teórica de lógica se asignó un martes a las 16:00 en el salón 103, querríamos que las otras clases que haya en la semana también se asignen a esta hora y salón.

Esta simetría podría lograrse en base a la especificación de las restricciones y preferencias para esta materia, de modo que al llegar a la solución final, si esta es lo suficientemente buena, cumplirá con las simetrías. Otra opción sería asegurar esta simetría siempre que esto fuera posible o conveniente. Podríamos, una vez que se asignó la primera clase de un grupo, asignar automáticamente las restantes al mismo horario y salón, es decir, asignar todo el ‘bloque’ de clases. Con esto sería más probable que se obtenga una asignación que preserve las simetrías, a la vez que se agilita el proceso de asignación al hacerlo más ‘determinístico’.

La asignación de un bloque podría evaluarse, y si no resulta conveniente dada la solución que se está construyendo, no aplicarla y dejar que el algoritmo siga su curso

normal, ya que en algún caso la aplicación de la asignación en bloque podría violar restricciones o preferencias de mayor peso. Por ejemplo, volviendo al caso de las clases de teórico de lógica, si entre las preferencias especificadas tenemos que se ha indicado un horario distinto para cada clase, no tiene sentido buscar una asignación ‘simétrica’ con todas las clases a la misma hora.

La asignación ‘en bloque’ podría aplicarse también a la asignación de prácticos inmediatamente que se han asignado los teóricos, de modo de asegurar que se dicten en el horario siguiente y/o en el mismo salón.

6 Implementación

6.1 Lenguaje de programación a utilizar

El algoritmo a implementar deberá realizar gran cantidad de cálculos, principalmente recorriendo estructuras de datos como vectores y matrices, y repetir este proceso varios cientos de veces hasta alcanzar una buena solución. Por lo tanto se hace necesario un lenguaje que permita crear y manejar eficientemente estas estructuras y que tenga gran potencia de cálculo.

Por otro lado, es necesaria una interfase que permita al usuario ingresar los datos en forma segura y eficiente, y le permita observar los resultados claramente.

Se consideró la portabilidad del programa, ya que podría ser ejecutado en distintas plataformas, por ejemplo bajo el sistema operativo Unix o bajo Windows.

Tomando en cuenta estos aspectos, se manejaron dos alternativas: El lenguaje C o Java, o sus versiones orientadas a objetos. El primero es, al momento, bastante más rápido que el segundo. A los efectos de una interfase amigable, Java ofrece grandes ventajas en cuanto a la facilidad de desarrollo. Ambos son altamente portables. Dado que era muy probable que encontrar una buena solución al problema demandara varias horas de ejecución, la rapidez fue una característica determinante en la elección del lenguaje, C. La interfase podría también desarrollarse como un módulo a parte, con el uso de bibliotecas de C, o bien en otro lenguaje apropiado, pero no es parte fundamental del trabajo propuesto. Se decidió utilizar **C++**, ya que la orientación a objetos proporciona y obliga a una codificación mucho más prolija y legible, mas adaptable a cambios y evoluciones , permite encapsular procedimientos, sin perder ninguna de las capacidades del C tradicional. Además se utilizaron sentencias correspondientes a la versión Ansi, para garantizar máxima portabilidad.

6.2 Estructuras de datos

Las principales entidades a representar en el problema son:

- la **estructura de la facultad**, con sus carreras, años, cursos, grupos y clases.
- la **"hormiga"**, quien se encargará de buscar la solución
- una estructura que permita almacenar los caminos seguidos por las hormigas, o sea los lugares por donde han dejado su **rastro**.
- la solución al problema, o sea la **asignación** de las clases a los salones y horarios.

Estas entidades se implementaron o bien como objetos, o bien como estructuras cuyos elementos son objetos. Como estructuras básicas se decidió utilizar vectores y matrices dinámicos, dado que se trata de estructuras estáticas en el caso de la facultad, y por simplicidad en la implementación. Por otra parte permiten un manejo simple y rapidez de acceso, siendo el mismo de orden 1 si nos manejamos con los índices de las mismas.

6.2.1 Estructura de la facultad

Se encuentra definida en el módulo “Estrfacu.hpp”
Sus elementos básicos son los objetos **salón** y **clase**.

6.2.1.1 Salón

El **salón** consta de:
- el identificador de salón.
- la capacidad del mismo (cantidad de estudiantes máxima)
- un vector de booleanos que indica qué “otras características” posee el mismo.

Los salones se agrupan en un vector, al cual se accede por su índice.

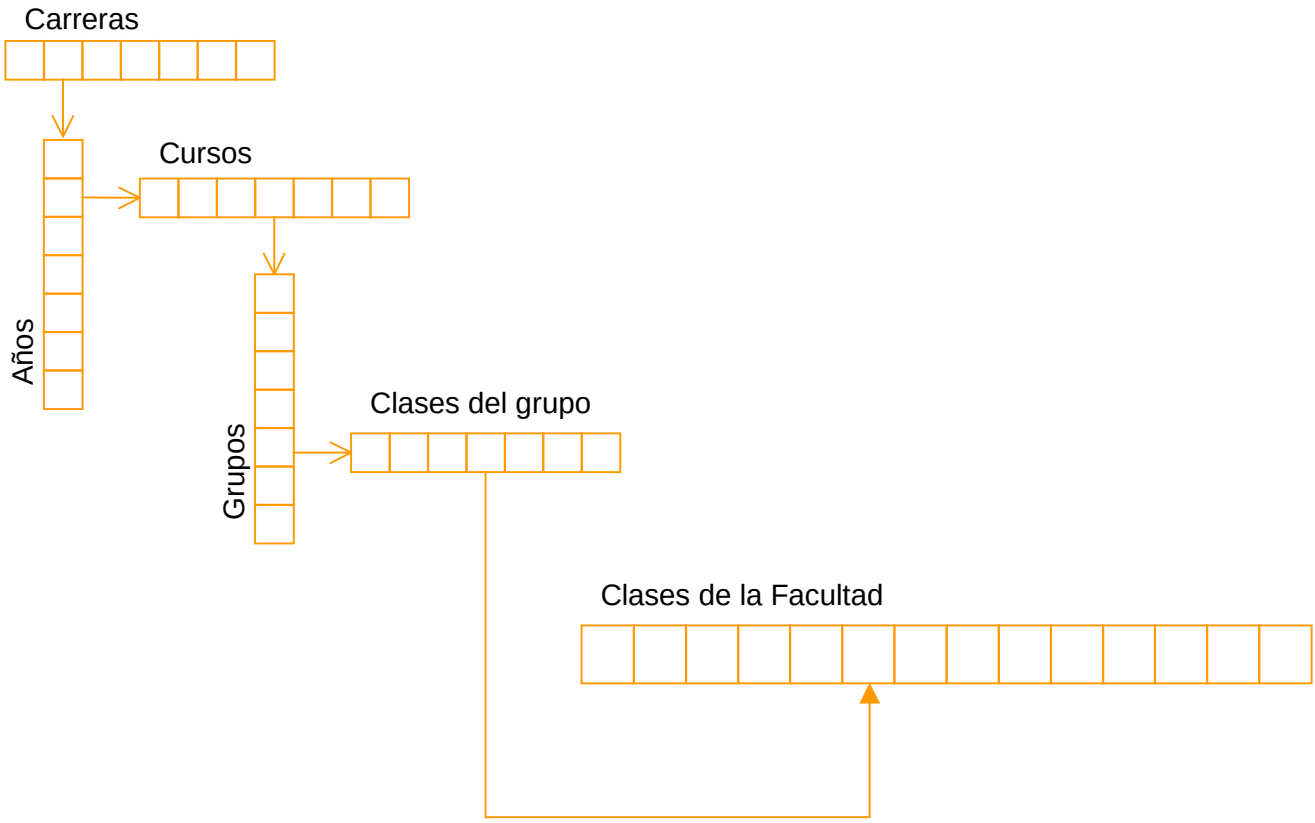
6.2.1.2 Clase

La **clase** consta de:
- el identificador de clase.
- la cantidad de estudiantes que asisten a ella.
- un número que indica su importancia.
- duración en ½ horas.
- un vector en el que pueden especificarse hasta 10 rangos horarios de preferencia.
- un vector de booleanos que indica “otras preferencias” de la clase, que determina su compatibilidad con un salón dado.
- un booleano que indica si la clase es teórica o no (práctica).
- un vector que contiene las “ubicaciones” de la clase, es decir en qué carreras, años, cursos y grupos se dicta.

Las clases se agrupan en un vector, al cual se accede por su índice, desde la estructura jerárquica de la facultad.

6.2.1.3 Estructura jerárquica de la facultad

Se implementa como una matriz de dimensión 5 (vectores de vectores), cuyas dimensiones son carrera, año, curso, grupo y clase.



Los vectores de carrera, año y curso tienen como elementos un producto cartesiano formado por un identificador y el vector de la categoría inmediatamente inferior. Los elementos de un vector de grupos agregan como datos el turno del grupo y un booleano que indica si es deseable que todas las clases del grupo sean asignadas con la misma hora de comienzo. Los elementos de los vectores de clases de un grupo solamente contienen el índice correspondiente a la clase en el vector de todas las clases de la facultad.

Esta organización se adoptó teniendo en cuenta que es altamente probable que una misma clase pueda formar parte de más de un grupo, curso, año o carrera, por lo tanto para evitar redundancia se decidió almacenar las clases en un vector a parte, que será accedido por su índice. Por otra parte, no es tan probable que ocurra lo mismo a nivel de curso, y mucho menos a nivel de año o carrera (es decir, que un curso o un año sea compartido por más de una carrera). La cantidad de información propia de una clase es mucho mayor que la correspondiente a un grupo, curso, año o

carrera. En base a esto, y por simplicidad, se prefirió manejar algo de redundancia en los datos a mantener estructuras separadas para cada nivel (curso, año, carrera).

6.2.1.4 Turnos

Un turno se especifica por un identificador, y un horario de comienzo y uno de fin dentro de un día, en forma de medias horas. Por ejemplo, si un día abarca un horario de 8:00 a 23:30, abarca 31 medias horas:
- la hora 8:00 corresponde a la media hora 0
- la hora 23:30 corresponde a la media hora 31.
- nota: observar que la última ½ hora **utilizable** es la 30, la cual **finaliza** a las 23:30.

Y si el primer turno va de 9:00 a.m. a 14:30 p.m., el mismo abarca 11 medias horas y se especifica con:
- horario de comienzo: 2
- horario de fin: 12
- nota: en este caso, observar que la última ½ hora utilizable en el turno es la 12, la cual corresponde a la hora 14:00 y **finaliza** a las 14:30.

Los turnos se almacenan en un vector con tope.

6.2.1.5 Perfiles

Un perfil es un conjunto de clases que no es deseable que se superpongan. Se implementa como un vector con tope.

6.2.2 La hormiga

6.2.2.1 Estructura

El objeto hormiga consta de los siguientes elementos:

```
class hormiga
{
     protected:
          vector_tope          lista_tabu;
          matriz2d_indice      lista_sh;
          matriz2d             asignacion;
          vector_sh            sh_clases;
          mat_horscaranio      hors_caranio;
.............
```

- **lista tabú:** vector de booleanos que indica las clases asignadas hasta el momento por la hormiga

- **lista_sh:** matriz de salones y horarios, que toma valores booleanos, y que será recorrida por la hormiga para determinar qué clase se asignará en un salón y horario dados. Una vez que una clase es asignada, se marcará en esta lista para no volver a utilizar el mismo salón y horario.
- **asignación:** matriz de salones y horarios, en la que se guarda la asignación realizada por la hormiga. Sus elementos son el índice de la clase asignada a ese salón y horario.
- **sh_clases:** vector que almacena para cada clase de la facultad el salón y horarios de comienzo y fin a los que se asignó una esa clase. Se utiliza en forma complementaria a la matriz de asignación, ya que provee un acceso directo por el índice de la clase para saber cómo se asignó la misma. Agrega redundancia pero permite evitar recorridas frecuentes y costosas para obtener datos de la asignación a la hora de evaluar la misma.
- **hors_caranio:** matriz de horarios, carreras y años cuyos elementos son listas de grupos. Indica para una carrera y año, qué horarios están ocupados por qué grupos. Se utiliza para evitar la superposición de clases que pertenecen a un mismo grupo. Cuando una clase de una determinada carrera y año se quiere asignar a un horario, se verifica primero que no exista ya en hors_caranio(horario, carrera, año) un grupo igual al de la clase a insertar; es decir, dados un horario, carrera y año, en el vector de grupos correspondiente no pueden haber elementos repetidos.

6.2.2.2 Trabajo de la hormiga

La hormiga se “mueve” recorriendo la lista_sh, y para cada salón y horario busca una clase a asignar que cumpla las restricciones necesarias, entre ellas, no debe estar en su lista tabú, y el grupo de la clase no debe estar ya insertado en hors_caranio. Luego que determina qué clase asignará (de acuerdo a la función “elegir_clase”), debe:

1. registrar las ternas de salón, horario y clase en la matriz de asignación.
2. insertar el salón y horario elegidos en sh_clases.
3. insertar el grupo de la clase en hors_caranio para cada grupo al que pertenezca la misma.
4. insertar la clase en la lista_tabú.
5. asignar en bloque (este procedimiento se explica en 8.3)
6. avanzar en su recorrida de la lista_sh.

Si no logró seleccionar ninguna clase para el salón y horario, debe ejecutar solamente el paso 6.

Luego que asignó todas las clases debe:
1. calcular el costo de la asignación que realizó.
2. dejar su rastro en la matriz temporal de rastros (mat_rastros_aux), donde se irá acumulando con el de las demás hormigas de la iteración actual.

6.2.3 Rastros

Los rastros dejados por las hormigas en su recorrido se almacenan en una matriz real de tres dimensiones (horarios, salones y clases). El valor del rastro dejado por una

hormiga depende de qué tan buena sea la asignación que realizó. La función básica por la que se calcula este rastro es:

rastro = mult_rastro / costo_asignación

donde **mult_rastro** es una constante que debe ser ajustada según los datos del problema en particular. Esto se debe a que el costo de la asignación depende en gran medida del tamaño del problema a resolver, de las características de los datos y de los pesos asociados a las preferencias.

Se han experimentado variantes de esta función para lograr un mejor resultado (ver sección 9.3.2.)

6.2.4 Asignación

La asignación se representa por una matriz de salones y horarios que contiene para cada (s, h):
- el índice de la clase asignada
- (-1) en caso que no se haya asignado ninguna clase.

A pesar de ocupar mucho espacio, esta forma de representar la asignación es eficiente en cuanto a inserciones y consultas.

El resultado final se despliega también por grupos, ya que esa es la forma utilizada en la facultad y que es de utilidad a los estudiantes. Para cada carrera, año y grupo se despliega una matriz donde las columnas son los días de la semana y las filas las horas del día (en ½ horas) y cada elemento es un identificador de clase y salón donde se asignó la misma.

6.3 Formato de los datos de entrada

Los datos del problema a resolver se agrupan en 4 archivos:
1. Clases
2. Salones
3. Facultad
4. Parámetros

nota: los nombres de los archivos no deben superar 8 caracteres más la extensión.

6.3.1 Archivo de clases

Contiene todas las clases dictadas en la facultad y todos sus datos:
La estructura del registro es:

cla ct0003_lu **est** 150 **imp** 10 **dur** 3 **teo** 1
ran 000 003 lu

otp 0 0 0 0 0 0 0 0 0 0

Donde:
- cla = identificador de la clase.
- est = la cantidad de estudiantes de la misma.
- imp = importancia
- dur = duración en ½ horas.
- teo = 1 si la clase es teórica
- 0 si es práctica.
- ran = rangos horarios de preferencia. Debe especificarse horario de comienzo y horario de fin (en ½ horas dentro de un día) y día de la semana. Pueden especificarse 0, 1 o varios rangos horarios de preferencia, repitiendo el renglón **ran**.
- otp = otras preferencias. Son diez posiciones booleanas. A cada posición puede asignársele un significado conveniente, que se relaciona con el identificador otp del archivo de salones. Por ejemplo, si definimos la primer posición como “tener enchufe” y la ponemos en 1, estaremos indicando que la clase requiere un salón que tenga enchufes. Al realizarse la asignación, se dará preferencia a salones que tengan un 1 en la primera posición del renglón otp, es decir, salones que tengan enchufe.

6.3.2 Archivo de salones

Contiene todos los salones de la facultad y sus datos.
La estructura del registro es:

sal 001 **cap** 060
otp 1 0 1 0 1 1 1 0 0 0

Donde:
- sal = identificador de salón.
- cap = capacidad máxima del salón.
- otp = otras características del salón. Se corresponde con otp en el archivo de clases.

6.3.3 Archivo de estructura de la facultad

Contiene la información de la facultad:

A. carreras, años, cursos, grupos y clases de grupos
B. turnos
C. cantidad de horas total
D. cantidad de horas por día.

A. carreras, años, cursos, grupos y clases de grupos:
La estructura del registro es:

car ca1 **ano** 01 **cur** cal0
gru h01_ **tur** 1 **eqh** 0
cla ct0003_lu
cla ct0003_mi
cla ct0003_ju

Donde:

- car = identificador de carrera
- ano = " " año
- cur = " " curso
- gru = " " grupo
- tur = turno del grupo
- eqh = 1 si es deseable que todas las clases del grupo comiencen a la misma hora
  0 si no.
- cla = identificador de clase perteneciente al grupo.

El identificador de grupo debe ser único dentro de una carrera y año. El grupo actúa a dos niveles: A nivel de curso, indica los grupos en los que el mismo está dividido, y las clases de un grupo pueden superponerse con las clases de otro grupo, pero no pueden superponerse las clases de un mismo grupo. A nivel de carrera y año, engloba las clases de distintos cursos, las cuales no pueden superponerse. Por ejemplo:

Sean:

car **ca1** ano **01** cur **cal0**
gru **h01**_ tur 1 eqh 0
cla ct0003_lu
cla ct0003_mi
cla ct0003_ju
cla cp0020_ma
cla cp0020_vi

gru **h02**_ tur 1 eqh 0
cla ct0003_lu
cla ct0003_mi
cla ct0003_ju
cla cp0020_ma
cla cp0020_vi

car **ca1** ano **01** cur **alg0**
gru **h01**_ tur 1 eqh 0
cla alt003_lu
cla alt003_mi
cla alp002_lu
cla alp002_mi

gru **h03**_ tur 1 eqh 0
cla alt003_lu
cla alt003_mi
cla alp002_lu
cla alp002_mi

car **ca2** ano 01 cur **eco0**
gru **h01**_ tur 1 eqh 0
cla ect003_lu
cla ect003_mi
cla ecp002_lu

gru **h03**_ tur 1 eqh 0
cla ect003_lu
cla ect003_mi
cla ecp005_vi

- Las clases pertenecientes a un grupo no pueden superponerse.
- Las clases pertenecientes al grupo **h01** pueden superponerse con las del **h02**, y las del **h03** con las del **h04**, ya que pertenecen a distintos grupos de un mismo curso.
- Las clases del curso de cálculo **cal0**, grupo **h01** no pueden superponerse con las del curso **alg0**, grupo **h01**, ya que tienen el mismo identificador de grupo, y ambos cursos pertenecen a la misma carrera (**ca1**) y año (**01**).
- Las clases del grupo **h02** pueden superponerse con las del **h03**.
- Las clases del curso **cal0**, grupo **h01** pueden superponerse con las del curso **eco0**, grupo **h01** ya que pertenecen a distintas carreras (**ca1** y **ca2** respectivamente).

La idea de un grupo que abarca varios cursos se aplica en la sección 9, al resolver la asignación para un juego de datos reales. A este tipo de grupos se le dio el nombre de **“maxigrupo”**.

B. Turnos

La estructura del registro es:

tur 1 000 012

Donde se indica:
- identificador de turno
- horario de comienzo (en ½ horas dentro de un día)
- horario de fin (en ½ horas dentro de un día)

C. cantidad de horas total:

cant_horas 155

D. cantidad de horas por día:

cant_horas_dia 31

6.3.4 Archivo de parámetros

Este archivo contiene los siguientes parámetros:

max_iter	máximo numero de iteraciones del algoritmo
cant_horm	cantidad de hormigas que participan en una iteración
exp_rastro	exponente del rastro en la función de cálculo de probabilidad
exp_visib	exponente de la visibilidad en la función de cálculo de probabilidad
coef_evap	coeficiente de evaporación del rastro.

Pesos asociados al cálculo de la visibilidad y del costo de la asignación:

peso_ran_hor	rangos horarios de preferencia.
peso_capac_salon	capacidad del salón
peso_otros_req	otros requerimientos
peso_desper_salon	desperdicio de salón
peso_espaciamiento	espaciamiento en días entre clases de un mismo grupo
peso_adyacencia	adyacencia horaria y de salón entre teóricos y prácticos de un mismo grupo.
peso_compactibilidad	compactibilidad de la asignación total

Otros parámetros utilizados por el algoritmo:

max_costo	máximo costo permitido en el cálculo de visibilidad y rastro
mult_rastro	factor que permite ajustar el valor del rastro dejado por la hormiga
min_visib	mínimo valor de visibilidad aceptado para una clase. Ver Variantes y mejoras al algoritmo básico.
max_visib	valor de la visibilidad utilizado cuando el costo es cero (dada la forma en que se calcula la visibilidad, un costo cero haría que la visibilidad fuera infinita)
min_prob	mínimo porcentaje que debe tener una clase sorteada en su valor de probabilidad con respecto al valor de probabilidad de la clase que está en mejores condiciones (tiene el mayor valor de probabilidad entre las clases candidatas) para ser asignada. Ver Variantes y mejoras al algoritmo básico.
min_rastro	mínimo rastro permitido en la matriz de rastros. Se utiliza también para inicializar la misma.

6.4 Formato de los datos de salida

Como resultado de la ejecución del programa se obtienen tres archivos de salida en formato texto:

- Salida.txt
- Fcostos.txt
- Frastros.txt

Salida.txt

En este archivo se obtiene una traza de la ejecución del algoritmo, sobre todo en lo que refiere a la carga de las estructuras de datos a partir de los archivos de entrada. Luego se presenta la solución final, en tres formatos. Primero se muestra la matriz de asignación, luego la solución día por día. A continuación, los costos desglosados de cada preferencia y el costo total de la asignación, con la iteración en la cual se alcanza el mismo. Finalmente se presenta la asignación grupo por grupo, formato utilizado en la asignación realizada manualmente por el IMERL (ver sección 9.1). En caso de que ocurriera algún error en la ejecución, será registrado en este archivo.

Fcostos.txt

Aquí se registra la evolución de los costos de la asignación a lo largo de las iteraciónes, divididos por costo de preferencia. Además incluye la evolución del costo total y las iteraciones a que corresponden. Este archivo es utilizado para graficar el costo de las soluciones por iteración.

Frastros.txt

Contiene la matriz de rastros. Como ésta es una matriz de tres dimensiones, se presenta la misma por clases, mostrando para cada una la matriz de dos dimensiones de salón y horario indicando el valor del rastro.

6.5 Principales elementos del algoritmo

6.5.1 Pseudocódigo del algoritmo principal

```
cargar cantidades para dimensionar estructuras de memoria que contendrán
los datos del problema a partir de archivos;

crear matriz de rastros (cant_salones, cant_hors, cant_clases);
crear matriz de grupos de una carrera y año (se utiliza para desplegar la
asignación por grupos);

cargar los datos del problema, a partir de los archivos, en las estructuras de
memoria;

crear la hormiga;
inicializar la matriz de rastros;

repetir
{
      inicializar la matriz de rastros auxiliar;
      para cada hormiga
      {
            repetir
            {
                  mover hormiga un paso (función “mover”);
            }
            hasta que se llene la lista tabú o la hormiga no pueda mover;

            si movio (la hormiga encontró una solución)
            {
                  calcular el costo de la asignación resultante;
                  si el costo es menor que el mejor hasta el momento
                  {
                        guardar la nueva solución;
                  }
                        almacenar rastro de la hormiga en matriz de rastros
                        temporal;
            }
            reinicializar la hormiga (su lista tabú, matriz de asignación, etc.);
      } fin para cada hormiga;

      si alguna hormiga llegó a una solución en esta iteración:
            agregar rastros dejados por las hormigas a la matriz de rastros
            definitiva y actualizarla según el coeficiente de evaporación

}
hasta condición de fin;

mostrar solución.
fin.
```

6.5.2 Pseudocódigo de la función “mover” que ejecuta la hormiga

Esta función devuelve 1 si la hormiga pudo moverse, y devuelve 0 si no pudo mover, es decir, no logró encontrar ninguna clase para asignar a ninguna de las parejas (s, h) que faltaban recorrer de la lista.

```
mover
{
    obtuve_clase = falso;
    fin_lista_sh = falso;
    mientras no obtuve_clase y no fin_lista_sh:
    {
        obtener siguiente pareja (s, h) libre en la lista_sh en la cual asignar;
        si no llegué al fin de la lista_sh
        {
            elegir una clase para la pareja (s, h);
            si no encontre una clase para (s, h)
            {
                si la lista_sh esta llena
                    fin_lista_sh = verdadero;
                si no
                    actualizar indices de lista_sh;
            }
            si no
            {
                encontre una clase candidata:
                obtuve_clase=TRUE;
            }
        }
    }
```

continuación:

```
    si fin_lista_sh
    {
        no hay espacio para assignar todas las clases:
        devolver(0) retornando;
    }

    Insertar la clase elegida en la lista_sh;
    Insertar la clase elegida en la matriz de asignación;

    Obtener ubicaciones de la clase;
    para cada ubicacion de la clase:
    {
        insertar la ubicación en matriz hors_caranio;
    }

    Insertar la clase elegida en el vector sh_clases;
    Insertar la clase elegida en la lista tabu de la hormiga;

    si la clase es teórica:
    {
        asignar en bloque las demás clases teóricas del grupo;
    }

    devolver(1);
}
fin mover
```

6.5.3 Pseudocódigo de la función “elegir clase”

Este procedimiento intenta encontrar una clase para una pareja de salón y horario (s, h). Para ello se construye una recta de probabilidades con las clases candidatas y se sortea una de ellas. Si la clase sorteada tenía un valor de prob menor que min_prob, se descarta la misma y no se asigna ninguna clase en la pareja (s, h).

```
Para cada clase de facultad  hacer
   Si cumple restricciones entonces
      Calcular prob = (rastro)^exp_rastro * (visb)^exp_visib ;
      Actualizar max_prob;
      Recta[clase] = prob;
   Si no
      Recta[clase] = 0;
Fin para

Si max_prob = 0 entonces
    no encontre clase
    fin;
Si no
   Para cada clase de facultad hacer
        Dividir recta[clase] por maxprob;
        // De esta manera se normaliza recta[clase] para que tome valores entre 0 y 1.
   fin para
Fin si

Armar la recta;
Sortear la clase;
Si la prob de la clase sorteada (recta[clase_sorteada]) es menor que min_prob
entonces
   la clase sorteada se descarta, como si no se hubiera encontrado ninguna clase
    para asignar (equivale a sortear la clase vacia);
si no
   devuelvo la clase sorteada;

fin elegir clase.
```

6.6 Decisiones de implementación

6.6.1 Hormigas

Uno de los puntos fundamentales del Ant System es la cooperación entre varias hormigas para encontrar un buen “camino”. Estas cooperan buscando caminos desde distintos puntos de partida, moviéndose independientemente y contribuyendo con su rastro, todas desplazándose en forma concurrente. El problema de asignación se ha diseñado de modo que todas las hormigas parten del mismo punto, es decir, recorren los horarios y salones en orden, y difieren sólo en la clase que eligen para cada pareja (s, h) dada y en la cantidad de rastro correspondiente a esa elección. Debido a esto, la única cooperación que existe es la que corresponde a la acumulación de los rastros de las distintas hormigas; no existe en tanto la cooperación por medio de distintos puntos de partida.

Si se pudiera disponer de múltiples procesadores, cada procesador ejecutando la tarea de una hormiga, donde compartan una memoria común se podría considerar la utilización de varias hormigas recorriendo a la vez el espacio de soluciones. Como no contamos con esa posibilidad la alternativa sería hacer un interleaving entre las distintas hormigas. Sin embargo como los rastros pueden impactarse sólo después de que cada hormiga termina de hacer su camino y como la longitud del camino es la misma para todas las hormigas ocurrirá que luego de completarse el camino de una hormiga, en el mismo ciclo van a completar sus caminos todas las hormigas del sistema. Recién entonces se podría actualizar el rastro, que afectará a posteriores hormigas.

De esta manera no es necesario implementar las hormigas como un vector que las contenga a todas, sino que basta con implementar la estructura para una sola hormiga, y utilizarla varias veces, simulando así la existencia de varias hormigas que recorren el espacio en paralelo.

Los rastros dejados por cada hormiga en una iteración se almacenan en una matriz de rastros temporal, sin que los rastros dejados por una hormiga influyan sobre las demás. Luego que todas las hormigas hicieron su recorrido, estos rastros se copian a una matriz de rastros definitiva, la cual sí será tomada en cuenta por las hormigas en la próxima iteración, al determinar qué camino seguir.

6.6.2 Restricciones y preferencias

Las restricciones y preferencias a considerar fueron especificadas en los puntos 3.2.1. y 3.2.2.
De las restricciones, algunas se cumplen en forma automática dada la forma en que se ha implementado el problema, y otras debieron ser implementadas. Existía un subconjunto de las restricciones que podían ser consideradas tanto como restricciones como preferencias. Se optó por considerar restricciones sólo aquellas que, de no cumplirse, no permitirían llegar a construir una solución en forma correcta. Las demás se consideraron como preferencias, y es posible poner mayor o menor énfasis en su cumplimiento ajustando distintos parámetros del algoritmo, entre ellos los pesos asociados a cada preferencia, min_prob, min_visib y max_visib.

Las restricciones que debe cumplir una clase para ser candidata a asignarse a un determinado salón y horario son:

- No debe existir en la lista tabú de la hormiga (de lo contrario esa clase ya ha sido asignada)
- Debe poder asignarse la totalidad de su duración sin que se superponga con otra clase ya asignada a ese salón.
- Debe poder asignarse la totalidad de su duración dentro de los límites horarios del día al que corresponde el horario.
- No debe superponerse ni en salón ni en horario con otras clases del mismo grupo.
- Debe asignarse dentro del turno especificado para su grupo.
- No pueden asignarse dos clases de un mismo grupo el mismo día, a no ser que una clase sea de teórico y la otra de práctico.
- No debe superponerse con otras clases de los perfiles a los que pertenezca (sin implementar)
- Debe poder disponer de los “recursos móviles” (sin implementar).

Las preferencias que se han considerado se dividen en dos grupos: las que se consideran para el cálculo de la visibilidad al asignar una clase, y las que permiten evaluar el costo de la asignación completa. Estas últimas incluyen a las anteriores. Para el cálculo de la visibilidad se han considerado preferencias que pueden evaluarse en forma local. Las mismas son:

- Rangos horarios de preferencia de la clase
- Capacidad del salón
- Otros requerimientos de la clase contra otras características del salón
- Desperdicio de la capacidad del salón.

Además de las anteriores, las preferencias que se consideran para evaluar el costo total de la asignación son:

- Compactibilidad
- Espaciamiento de las clases teóricas de un grupo en los días de la semana
- Adyacencia horaria y de salón entre clases teóricas y prácticas

6.6.3 Elección de una clase

Cálculo de la visibilidad

Para el cálculo de la visibilidad se consideran las preferencias indicadas en el punto anterior, ponderadas cada una por el peso correspondiente (el mismo que se utiliza en el cálculo del costo de la asignación, aunque podría ser distinto) El costo de asignar una clase a un salón y horario es:

costo(salón, horario, clase) = peso1 * preferencia1 + peso2 * preferencia2 + ...

Luego la visibilidad se calcula como:

Si costo $\neq 0$
    visibilidad = 1/costo;
Si costo = 0
    visibilidad = max_visib;

Cálculo de la probabilidad

Se buscaron maneras de aportar más inteligencia al algoritmo en cuanto a la elección de una clase. Dada una pareja (s, h) queremos sortear una clase que no solo cumpla con las restricciones, sino que también satisfaga un conjunto de preferencias locales ya que cuanto mejores decisiones locales tome más probabilidad tendrá la hormiga de arribar a una mejor solución.

La probabilidad que se asigna a cada clase candidata para un salón y horario se calcula en base a la matriz de rastros y a la visibilidad de la clase.

Si la clase candidata no tiene una visibilidad suficientemente buena, se le asigna visibilidad cero. Esto equivale a no considerar la clase como candidata; de alguna manera intentamos depurar el espacio de clases que pueden ser sorteadas:

si (visib < min_visib)
    visib = 0;

La probabilidad asignada a cada clase candidata para el sorteo de clase se calcula por la fórmula típica de Ant System:

$prob = (rastro)^{exp\_rastro} \times (visib)^{exp\_visib}$

Con este valor de prob, normalizado entre 0 y 1, se construye una recta a partir de la cual será sorteada la clase. La probabilidad que una clase tiene para ser sorteada es igual al valor de prob de la misma dividido entre la suma de los valores de prob correspondientes a todas las clases candidatas.

Luego de sorteada la clase, si el valor de prob correspondiente a la misma es menor que el parámetro min_prob, entonces se descarta la clase y no se asigna ninguna clase a la pareja (s, h).

6.6.4 Costo de la asignación

El costo de la asignación se calcula en la misma forma que el costo por visibilidad, pero teniendo en cuenta todas las preferencias, es decir:

costo(asignación) = peso1 * preferencia1(asignación) +
                    peso2 * preferencia2(asignacion) + ...

6.6.5 Rastros y evaporación

El que un rastro en una terna (s, h, c) llegue a cero significa para la hormiga que ese "camino" deja de existir, ya que la probabilidad resultante para esa clase 'c' será cero siempre, dada la fórmula utilizada para el cálculo de la probabilidad:

prob = visibilidad(s, h, c)$^{exp\_visib}$ * rastro(s, h, c)$^{exp\_rastro}$

Debido a esto se debe mantener un valor mínimo para todos los rastros, especificado por el parámetro **min_rastro**.

Los rastros son evaporados de acuerdo al coeficiente de evaporación **coef_evap**, pero nunca descienden del mínimo especificado en **min_rastro**.

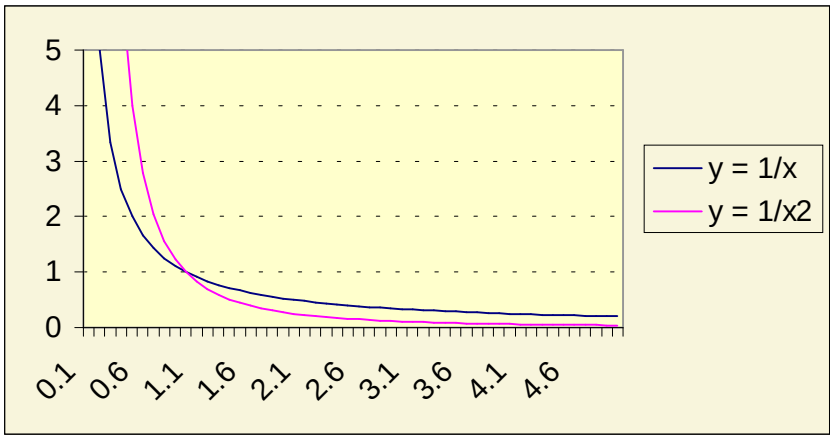
Para el cálculo del valor del rastro de acuerdo al recorrido efectuado por la hormiga, se utilizó en primera instancia la fórmula indicada en el punto 6.2.3., que considera el costo de la asignación realizada:

rastro = mult_rastro / costo_asignación

Luego se vio la necesidad de guiar a las hormigas por rastros que guardaran una mejor relación con la solución obtenida. Para ello se decidió usar la siguiente fórmula:

rastro = mult_rastro / (costo_asignación)^2

la cual, para buenas soluciones (costo_asignación bajo), devuelve un valor mayor de rastro, y para malas soluciones (costo_asignación alto), devuelve un valor menor de rastro, según puede apreciarse en la siguiente gráfica:



7 Pruebas(I) – Juego de datos ficticio

7.1 Juego de datos de prueba ficticios

Para las primeras pruebas se confeccionó un juego de datos con una carrera compuesta por dos años, uno de ellos con 5 cursos y el otro con 2. Cada curso tiene a lo sumo 2 grupos, y cada grupo tiene a lo sumo 3 clases. El total de clases es 17, las cuales deben ubicarse en 3 salones. La cantidad total de ½ horas es 40, divididas en 4 días de 10 horas cada uno. Se consideraron 2 turnos, que dividen al día en 2 mitades. El juego de datos completo está en el Apéndice A.

Se consideraron uno o dos rangos horarios de preferencia para cada clase, apenas mayores que la duración de la misma.

Los grupos tienen iguales identificadores en distintos cursos, pero se consideraron como grupos distintos. Al momento no se consideró la idea de grupos que abarcaran más de un curso. Ver variantes y mejoras al algoritmo básico.

7.2 Pruebas y resultados

De acuerdo a los datos utilizados, una muy buena asignación, sobre todo contemplando las preferencias de rangos horarios y las relativas a capacidad y desperdicio de espacio en salones sería:

	dia 0										dia 1									
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
sal.0 (160)	cla 0		cla 1				cla 2				cla 3			cla 4			cla 6			
sal.1 (90)	cla 11			cla 12			cla 16							cla 14						
sal 2 (160)																				

	dia 2										dia 3									
	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
sal.0 (160)						cla 7					cla 3			cla 5					cla 8	
sal.1 (90)	cla 13					cla 9							cla 15		cla 10					
sal 2 (160)																				

A continuación se presenta un resumen de las pruebas efectuadas y se comentan algunos resultados. Algunas de las pruebas realizadas pueden verse en el diskette que acompaña este informe.

Resumen de pruebas (A)

Nro. de Prueba	Nro. de iteraciones	Coeficiente de evaporación	Exponente del rastro	Iteración de la mejor solución	Mejor solución	Planilla excel corresp.	Observaciones
A1	1000	0.99	0.2	238	11.87	02	Los rastros llegan a 0 debido a la evaporación.
A2	1000	0.999	0.2		14.67	03	Se evita que los rastros lleguen a 0. No se ve buena evolución en las soluciones.
A3	2000	0.999	0.2	192	14.4		
A4	2000	0.999	0.2	972	11.17		
A5	6000	0.999	0.2	2518	8.44	04	
A6	12000	0.9995	0.2		7.7		
A7	18000	0.9995	0.2	9078	6.64	09	Hay cierta evolución en las soluciones, pero se mantiene gran aleatoriedad.
A8	8000	0.9995	0.3	6760	7.04	06	Hay clara evolución en las soluciones.
A9	8000	0.9995	0.4	6433	6.28	07	Las soluciones evolucionan más rápidamente a valores menores.
A10	8000	0.9995	0.5	7064	6.54	08	Idem anterior.
A11	8000	0.9995	0.5	3723	6.54		Idem anterior.
A12	8000	0.9995	0.7	4757	5.21	10	Idem anterior.
A13	8000	0.9995	0.9	1883	5.65	11	Idem anterior.
A14	4000	0.9995	0.9	3154	5.21	12	Idem anterior.
A15	4000	0.9995	2	1391	5.69	13	Buenas soluciones entre iteraciones 1000 a 2000. Luego depende fuertemente de los rastros y tiende a soluciones de costo 40, sin explorar.
A16	4000	0.9995	2	1095	5.71	14	Idem anterior.

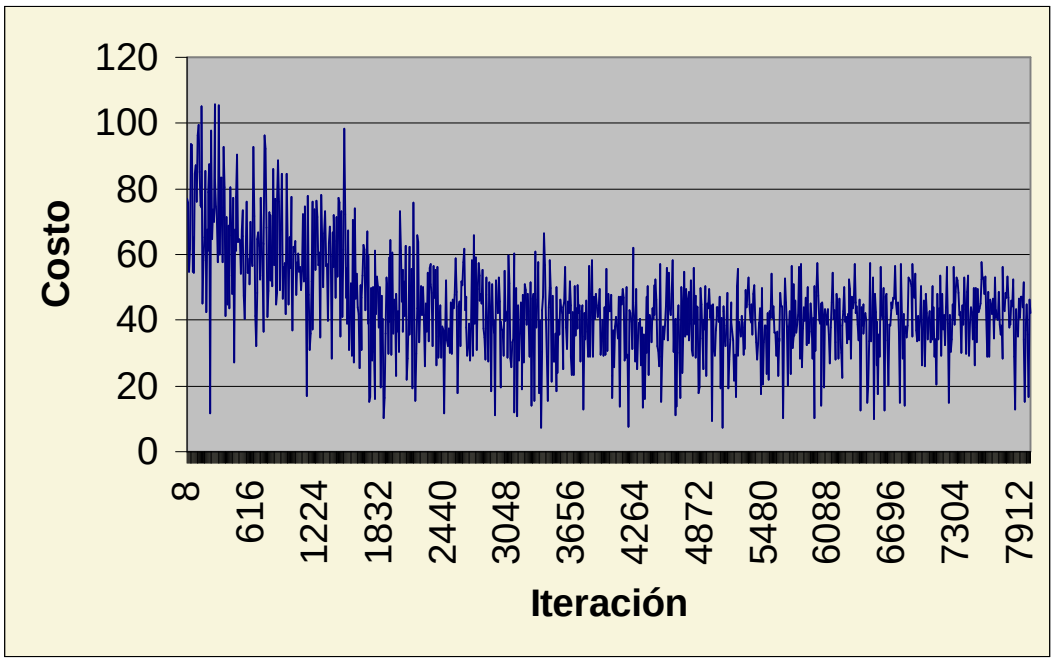
Prueba A12:
El siguiente resultado se logró con los siguientes parámetros:

max_iter	8000	peso_ran_hor	0.5
cant_horm	5	peso_capac_salon	0.3
exp_rastro	0.7	peso_otros_req	0.17
exp_visib	0.2	peso_desper_salon	0.66
coef_evap	0.9995	peso_espaciamiento	0.17
max_costo	400	peso_adyacencia	0.1
mult_rastro	2	peso_compactibilidad	0.1
min_visib	0		
max_visib	32000		

	I 0	I 1	I 2	I 3	I 4	I 5	I 6	I 7	I 8	I 9	I10	I11	I12	I13	I14	I15	I16	I17	I18	I19
salón 0	0	0	1	1	-1	2	2	2	-1	-1	3	3	3	4	4	-1	6	6	6	-1
salón 1	11	11	11	12	12	-1	16	16	16	16	14	14	14	-1	-1	9	9	-1	-1	-1
salón 2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

	I20	I21	I22	I23	I24	I25	I26	I27	I28	I29	I30	I31	I32	I33	I34	I35	I36	I37	I38	I39
salón 0	-1	13	13	13	-1	7	7	7	-1	-1	15	15	15	5	5	8	8	10	10	-1
salón 1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
salón 2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Evolución del costo de la solución:



En esta primera etapa de pruebas se lograron resultados aceptables, y se buscó solucionar algunos inconvenientes. Se detectó que los rastros dejados por las hormigas llegaban a cero debido a la gran cantidad de iteraciones, aún cuando fueran inicializados en valores altos. Dependiendo de la cantidad de iteraciones, se debió utilizar un coeficiente de evaporación de 0.999 o 0.9995 para evitar llegar a rastros con valor cero. Notar que $(0.99)^{1000} = 4.3 \times 10^{-5}$, y que $(0.999)^{1000} = 0.36$.

Con un exponente de rastro de 0.2, fue necesaria una gran cantidad de iteraciones para detectar una mejora en la solución hallada. Aumentando este exponente, es decir aumentando la importancia del rastro al sortear una clase, se logró alcanzar mejores soluciones en menos cantidad de iteraciones (pruebas A8 a A16). Sin embargo, luego de un cierto número de iteraciones, las soluciones tienden a ser construidas cada vez más en base a los rastros, los cuales se han ido acumulando poco a poco, y el algoritmo tiende a converger a soluciones mediocres, de costo medio. Se observa que esto ocurre debido a que este tipo de solución se obtiene gran cantidad de veces, acumulando así más rastro, en cambio las soluciones de bajo costo se alcanzan pocas veces, y no logran acumular suficiente rastro como para prosperar.

Otro problema detectado, y evidente en la solución obtenida en A12, es la “ansiedad” del algoritmo por asignar clases en cada pareja (s, h) de su recorrido. Esto produce una asignación bastante compacta, pero a cambio, las últimas clases se asignan en horarios y salones que no son los más adecuados.

Soluciones propuestas:
En el siguiente conjunto de pruebas, se aumenta el exponente de la visibilidad en una primera etapa, y luego se aumenta el peso de la preferencia “desperdicio de la capacidad del salón”, de modo de lograr asignar las clases en salones acordes a su cantidad de estudiantes.

Resumen de pruebas (B)

Nro. de prueba	Nro. de iteraciones	Coeficiente de evaporación	Exponente del rastro	Exponente de la visibilidad	Iteración de la mejor solución	Mejor solución	Planilla excel corresp.	Observaciones
B1	4000	0.9995	0.7	0.2	3180	6.01	15	
B2	4000	0.9995	0.7	0.5	1217	5.21	16	Se observan costos en un menor rango (costo promedio 30) y muchas soluciones cercanas a 10 en toda la prueba.
B3	4000	0.9995	0.7	0.5	1815	5.11	17	Idem anterior.
B4	4000	0.9995	0.7	0.9	718	5.48	18	
B5	4000	0.9995	0.7	0.9	258	4.95	19	
B6	400	0.9995	0.7	0.9	69	6.47	20	Con exp. Rastro > 0.2 no se observan mejoras en la solución (aprendizaje en base a rastros). Se obtienen buenas soluciones desde el comienzo.
B7	4000	0.9995	0.7	0.2	3869	7.37		CAMBIO: Peso desp. Salón: 1.00. Se busca obtener un mejor uso de los salones.
B8	4000	0.9995	0.7	0.2		5.38		CAMBIO: Peso desp. Salón: 0.66
B9	4000	0.9995	0.7	0.2	3743	14.01		CAMBIO: Peso desp. Salón: 2.0
B10	4000	0.9995	0.7	0.2	3617	34.51		CAMBIO: Peso desp. Salón: 5.0. No se observan mejoras en la solución.
B11	6000	0.9995	0.7	0.2	5186	29.21		Buena asignación, pero todavía hay un mal uso de los salones. Costo desperdicio de salón: 25.00.
B12	4000	0.9995	0.7	0.2	2614	46.51	21	Costo desperdicio de salón: 10.00.
B13	4000	0.9995	0.7	0.2	1056	44.91		
B14	8000	0.9995	0.7	0.2	3009	49.74		
B15	6000	0.9995	0.7	0.2	4497	30.44	22	CAMBIO: min_prob: 0.7. Peso desp. Salón: 10.00.
B16	6000	0.9995	2		2202	20.94	23	CAMBIO: min_prob: 0.85. No se observa evolución. Las mejores soluciones se dan al comienzo.

El aumento del exponente de la visibilidad hace que se obtengan mejores soluciones, y mayor cantidad de buenas soluciones. El costo promedio de las mismas bajo, y son

necesarias menos iteraciones para obtener soluciones de la misma calidad. Esto se ve en las pruebas B1 a B6. Con un exponente de rastro no tan alto (0.7) y un exponente para la visibilidad mayor (0.5) se logra un valor promedio de la solución mejor (aprox. 30). Aún así no se logra una mejora en cuanto a la utilización de los salones. A partir de la prueba B7 hasta la B14 se aumenta el peso asociado a la preferencia “desperdicio de la capacidad del salón”, buscando una mejora en ese aspecto. Observar que el costo de la solución aumenta como consecuencia de la variación en el peso de la preferencia, y no porque la calidad de la solución empeore. No se obtienen mejoras sustanciales en cuanto a la utilización de los salones, ya que el algoritmo sigue siendo muy ansioso.

En las pruebas B15 y B16 y en las siguientes pruebas C, se comienza a utilizar el parámetro “min_prob” para intentar eliminar la ansiedad. Ver el punto 3.1 para una descripción del funcionamiento del mismo.

Resumen de pruebas (C):

Nro. de Prueba	Nro. de iteraciones	Coeficiente de evaporación	Exponente del rastro	Exponente de la visibilidad	Min_prob	Iteración de la mejor solución	Mejor solución	Planilla excel corresp.	Observaciones
C1	4000	0.9995	0.5	0.2	0.85	1174	18.61	25	
C2	8000	0.9995	0.5	0.2	0.85	1174	22.07	26	Buenas soluciones solo al principio. No evoluciona.
C3	4000	0.9995	0.5	0.5	0.85	529	33.11	27	
C4	4000	0.9995	0.7	0.1	0.85	2596	22.82	28	
C5	4000	0.9995	0.7	0.9	0.85	3477	21.04	29	
C6	4000	0.9995	0.9	0.05	0.85	3302	31.71	30	
C7	8000	0.9995	0.9	0.05	0.85	4758	23.24	31	La solución converge a un costo de 40.
C8	6000	0.9995	0.9	0.05	0.95	1735	15.34	32	
C9	10000	0.9995	0.7	0.05	0.95	6573	16.71	33	Se enlentece la convergencia pero sigue convergiendo a 40.
C10	6000	0.9995	0.9	0.2	0.95	900	16.74	34	Continua convergiendo a 40, pero más rápidamente pues se aumentó el exponente de la visibilidad.

Con la utilización de min_prob se logra reducir el costo de la asignación de aproximadamente 45 (antes de utilizarlo) a 16. Se estudian nuevamente valores apropiados para el exponente del rastro y la visibilidad, y se observa que la combinación utilizada antes de usar min_prob sigue siendo la mejor. Con min_prob se logra disminuir la ansiedad del algoritmo al asignar, y por lo tanto se logra un mejor uso de los salones y horarios, como se observa en el resultado de la prueba C10:

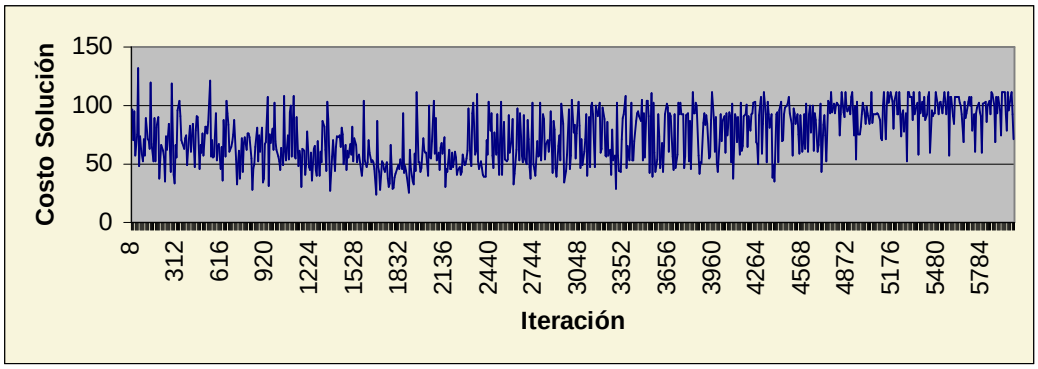
	dia 0										dia 1									
	I 0	I 1	I 2	I 3	I 4	I 5	I 6	I 7	I 8	I 9	I10	I11	I12	I13	I14	I15	I16	I17	I18	I19
salón 0	0	0	1	1	-1	2	2	2	-1	-1	3	3	3	5	5	-1	6	6	6	-1
salón 1	11	11	11	12	12	-1	16	16	16	16	14	14	14	-1	-1	-1	9	9	-1	-1
salón 2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

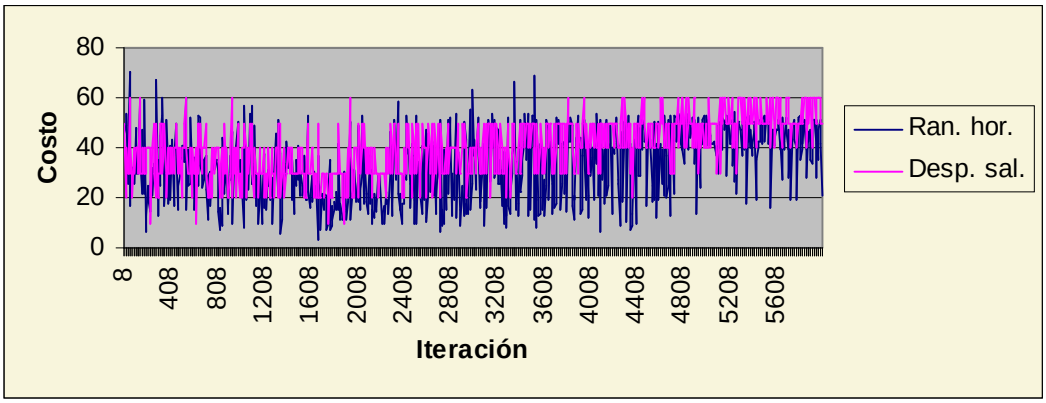
	dia 2										dia 3									
	I20	I21	I22	I23	I24	I25	I26	I27	I28	I29	I30	I31	I32	I33	I34	I35	I36	I37	I38	I39
salón 0	-1	-1	-1	-1	-1	7	7	7	-1	-1	-1	-1	-1	4	4	-1	8	8	-1	-1
salón 1	13	13	13	-1	-1	10	10	-1	-1	-1	15	15	15	-1	-1	-1	-1	-1	-1	-1
salón 2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Costos obtenidos:
costo de la solución: 16.74
costo_rh es : 6.0000
costo_cap es : 0.0000
costo_desp es : 10.0000
costo_oreq es : 0.0000
costo_comp es : 0.0000
costo_espac es : 0.3400
costo_ady es : 0.4000

En esta solución las clases 10, 13 y 15, con 78, 80 y 90 estudiantes respectivamente, quedan asignadas en el salón 1, que tiene menor capacidad que el cero. Esto no ocurría en la prueba A12. El costo de desperdicio de salón es el mínimo (no puede ser cero dadas la cantidad de estudiantes de las clases y la capacidad de los salones, siempre habrá un desperdicio mínimo). El costo por rangos horarios desfasados se debe a la clase 14 que está un poco desplazada con respecto a su rango horario de preferencia, y a la clase 10, que se asignó en un día distinto al preferido.

Evolución del costo de la solución:





El costo de la solución decrece hasta aproximadamente la iteración 1700, y luego aumenta, guiado por rastros que se han ido acumulando y pertenecen a soluciones de costo medio. El algoritmo tiene entonces menos posibilidades de explorar.

Debe buscarse una mejora que permita mantener los rastros que corresponden a las buenas soluciones, y que no acumule rastros correspondientes a soluciones de costo medio. Con este objetivo, se hicieron las siguientes modificaciones:

- Se permite una mayor evaporación de los rastros, de modo que el algoritmo pueda, a lo largo de muchas iteraciones, seguir renovándolos, y que no se produzca una acumulación demasiado alta en lugares mediocres. No se llega a un rastro cero gracias a la utilización de min_rastro.
- Se modifica la función de cálculo del rastro elevando el costo al cuadrado:

  $$\text{rastro} = \text{mult\_rastro} / (\text{costo})^2$$

De esta manera se intenta incrementar o “premiar” los rastros correspondientes a buenas soluciones, ya que a menor costo, se obtendrá un mayor rastro. Los resultados se ven en el siguiente resumen de pruebas:

Resumen de pruebas (D):

Nro. de prueba	Nro. de iteraciones	Coeficiente de evaporación	Exponente del rastro	Cantidad de hormigas	Mult_rastro	Min_rastro	Iteración de la mejor solución	Mejor solución	Observaciones
D1	2000	0.7	0.9	15	100	0.1	404	17.14	
D2	1000	0.8	1	10	100	1	924	12.84	
D3	1000	0.8	1	10	500	1	487	15.57	
D4	1000	0.9	1	10	500	1	253	18	Rastros crecen muy poco y se evaporan muy pronto. No hay evolución en las soluciones. Valor de la sol. Promedio: 75
D5	1500	0.7	0.9	10	500	0.1	1224	33.28	Mayor aumento en los rastros. Rastros poco frecuentes se evaporan, se mantienen sólo rastros más frecuentes. Sol. Promedio: 85.
D6	1500	0.7	0.9	10	5000	0.1	1	41.18	Rastros más altos en lugares más frecuentes (aprox. 15.00). Costo de la solución casi cte. Promedio: 105.
D7	1500	0.7	0.9	10	5000	1			Rastro promedio: 20.00.
D8	1500	0.7	0.9	10	5000	1	587	32.68	
D9	1500	0.8	0.9	10	5000	1	174	33.18	
D10	1000	0.95	1	10	100	1	288	32.18	Min_prob = 0.8, Exp_visib = 0.1. Mal resultado. No se ve utilidad de los rastros.

En estas pruebas, se probaron principalmente distintos valores para el coeficiente de evaporación y para el parámetro mult_rastro. Mult_rastro debió aumentarse significativamente desde las pruebas “C”, ya que de lo contrario los rastros se perdían totalmente debido a la mayor evaporación. Por otra parte, si mult_rastro está muy alto, no conduce a buenas soluciones, ya que las hormigas son guiadas desde temprano por los rastros, los cuales no corresponden a buenas soluciones. La evaporación más rápida permite al algoritmo llegar a buenas soluciones, pero no se observan grandes mejoras.

Resumen de pruebas (E):

Nro. de prueba	Nro. de iteraciones	Coeficiente de evaporación	Exponente del rastro	Exponente de la visibilidad	Mult_rastro	Iteración de la mejor solución	Mejor solución	Observaciones
E1	500	0.95	1	0.1	500	1	46.61	Rastros y solución final no coinciden. Costo de la solución tiende a 100.00.
E2	500	0.95	0.5	0.5	500	1	23.64	
E3	500	0.95	0.5	0.5	500	3	27.07	Max_visib = 5
E4	500	0.95	0.2	0.8	500	141	23.24	Probabilidad fuertemente guiada por buenos rastros.
E5	1000	0.95	0.2	0.8	500	394	15.17	
E6	1000	0.95	0.2	0.8	100	315	15.41	
E7	1000	0.95	0.1	0.9	100	901	19.51	

Se utilizó min_prob = 0.8.

En estas pruebas, se parte de un valor medio de mult_rastro y un coeficiente de evaporación alto (poca evaporación). El exponente del rastro se va disminuyendo y el de la visibilidad aumentando, hasta lograr buenas soluciones.

Esto muestra que los distintos parámetros del algoritmo están muy interrelacionados, y pueden combinarse de distintas formas para alcanzar un mismo resultado.

Otro problema que se detecta está relacionado con el parámetro max_visib, o sea la máxima visibilidad que una clase puede tener. Este máximo se alcanza cuando se encuentra una clase cuyo costo de asignación a un salón y horario es cero. Si max_visib contiene un valor muy elevado, y el exponente de la visibilidad es también alto, una clase de costo cero tendrá una probabilidad altísima de ser asignada, sin que los rastros tengan ninguna participación. Esto no produce buenos resultados, ya que elimina la posibilidad de “exploración” o aleatoriedad del algoritmo. Por lo tanto, en la prueba E3 se corrige el valor de max_visib, pasando de 32000 a 5. Este cambio permite una mayor variedad en las soluciones encontradas y una mejor evolución.

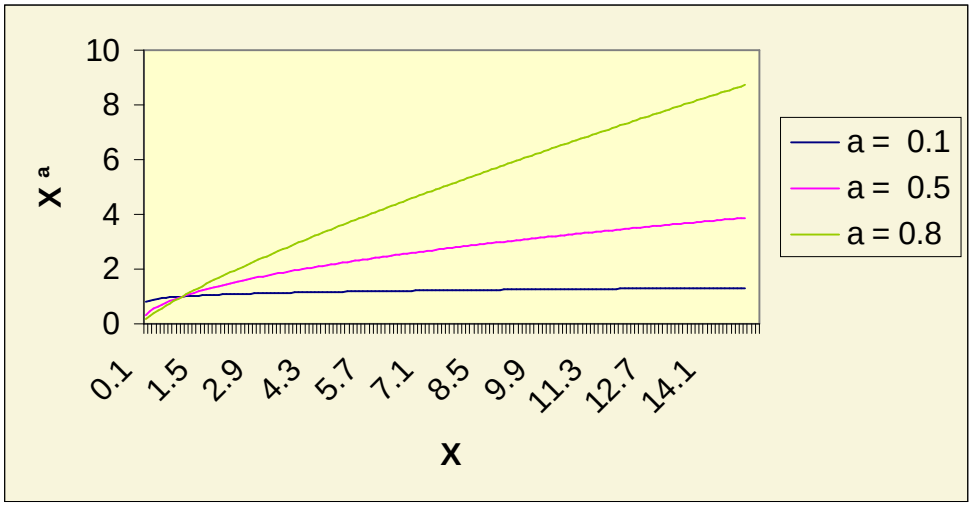
Solución encontrada en prueba E7:

	dia 0										dia 1									
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
salón 0	0	0	1	1	-1	-1	2	2	2	-1	-1	3	3	3	-1	6	6	6	-1	-1
salón 1	11	11	11	12	12	16	16	16	16	-1	13	13	13	-1	-1	10	10	-1	-1	-1
salón 2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

	dia 2										dia 3									
	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
salón 0	-1	-1	-1	5	5	7	7	7	8	8	-1	-1	4	4	-1	-1	-1	-1	-1	-1
salón 1	14	14	14	-1	-1	9	9	-1	-1	-1	15	15	15	-1	-1	-1	-1	-1	-1	-1
salón 2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

7.3 Ajuste de parámetros

Exponentes exp_rastro y exp_visib:
Estos exponentes determinan la importancia relativa del rastro y la visibilidad en la formula de cálculo de probabilidad, pero además determinan el grado en que influye la variación del rastro y la visibilidad. Cuanto menor sea el exponente, menor será la variación del resultado de elevar el valor (rastro o visibilidad) a ese exponente. Este comportamiento se advierte al observar las siguientes gráficas:



Pesos asociados a cada preferencia:
El valor de los pesos asociados a cada preferencia determina el valor resultante de la visibilidad de una clase, la cual se calcula en la forma vista en el punto 6.5.3. Esto

implica que a mayores pesos, mayor será el costo resultante y menor el valor de la visibilidad; esto repercute en el cálculo de la probabilidad (ver punto 6.5.3.).

Evaporación y rastros

El **coeficiente de evaporación** del rastro debió ser fijado en un número bastante alto, es decir, muy cercano a 1, ya que es necesario un gran número de iteraciones para alcanzar una buena solución. Si el coeficiente de evaporación no es suficientemente alto, la gran cantidad de iteraciones hace que el rastro se evapore demasiado rápidamente, y no puede entonces ser utilizado por las hormigas al construir su camino. Otra razón por la que el coeficiente de evaporación debe ser alto en relación al utilizado en otros problemas resueltos con Ant Systems es la siguiente: en el TSP, en donde las aristas son caminos entre ciudades, una arista cualquiera tiene mucha más probabilidad de ser recorrida por una hormiga que una terna de salón, horario y clase ser seleccionada para formar parte de una solución en el problema de asignación de salones y horarios a cursos. Esto se debe a que en el problema de asignación no se recorre un grafo, sino un hipergrafo, lo cual genera un número mucho mayor de combinaciones posibles. En el primer caso las aristas son recorridas más veces, entonces el coeficiente de evaporación puede ser menor.

La fórmula por la cual se actualiza la matriz de rastros al final de cada iteración del algoritmo es la siguiente:

rastro = rastro * coef. evaporación + nuevo rastro

Si para una pareja (s, h) no se ha asignado una clase, el nuevo rastro vale cero, y solamente se evaporará el rastro ya existente. En este caso entonces, luego de por ejemplo 1000 iteraciones, tendremos:

rastro = rastro * $(\text{coef. evaporación})^{1000}$.

Sea por ejemplo coef. evaporación = 0.9.

Entonces $(0.9)^{1000} = 1.74 * 10^{-46}$, lo cual lleva el rastro rápidamente a su valor mínimo.

Si el coef. de evaporación fuera 0.995:

$(0.995)^{1000} = 6.65 * 10^{-3}$, número mucho mayor que el anterior, lo que permite preservar un poco más los rastros con el correr de las iteraciones del algoritmo.

Otro parámetro que influye en la evolución de los rastros es **mult_rastro**, ponderador que determina la cantidad de rastro dejada, de acuerdo a la fórmula de cálculo del rastro vista en 6.5.5.

La combinación de **mult_rastro** y **coef_evap** determina cuánto aumentarán los rastros, cuánto se evaporarán, y cuántas iteraciones subsistirán sin nuevos aportes de rastro.

Valor inicial del rastro:
El valor con el que se inicializa la matriz de rastros influye en la importancia relativa que tendrá el rastro que aporten las hormigas en su recorrido. Por ejemplo, si el rastro inicial es 5 y una hormiga aporta en promedio un rastro de 5 para cada clase asignada, el rastro para esa clase se verá incrementado casi en un 100% (menos la cantidad evaporada). Si en cambio el rastro inicial fuera 20, agregarle un rastro de

valor 5 significa tan sólo un 25% de aumento (menos la evaporación). Esta diferencia relativa entre un rastro incrementado y otro que no se incrementó influirá al momento de
asignar probabilidades a las clases candidatas para una pareja (salón, horario); en el primer caso, el rastro tendrá más importancia que en el segundo.

7.4 Conclusiones extraídas del estudio de las pruebas con el juego de datos ficticio

En este conjunto de pruebas se obtuvieron buenas soluciones, y se lograron mejoras a partir de las modificaciones a la función de rastros y a partir de la introducción del parámetro min_prob. Puede observarse que realizando nuevos cambios al algoritmo básico, se lograrían resultados más interesantes.

Un problema que se detectó es la dificultad para calibrar los distintos parámetros, ya que los mismos se hallan muy interrelacionados.

Debido al diseño elegido, se hace necesario controlar la 'ansiedad' del algoritmo por asignar siempre alguna clase a una pareja de salón y horario. Ver punto 5.2.2. El parámetro min_prob permite un control bastante efectivo de esta característica.

Un exponente de visibilidad muy alto influye demasiado en las primeras iteraciones y provoca una convergencia rápida a un óptimo local. Lo mismo sucede si se utiliza un exponente de rastro alto o si el incremento de los rastros es muy elevado en cada paso.

El coeficiente de evaporación debe ser lo suficientemente alto como para evitar la rápida evaporación de rastros, debida a la gran cantidad de iteraciones necesarias. La amplitud del espacio de soluciones hace que muchas ternas potencialmente interesantes no sean recorridas con la suficiente frecuencia.

El número de hormigas no parece tener gran influencia en los resultados obtenidos.

8 Variantes y mejoras al algoritmo básico

Dados los resultados obtenidos y los problemas detectados en la sección anterior, se estudia la implementación de mejoras que involucran la creación de nuevos parámetros que permitan regular mejor el comportamiento del algoritmo; también se implementa una variante del Ant System que llamaremos “Mejor Hormiga”. Finalmente se estudia la “asignación en bloque” de clases de un mismo grupo.

8.1 Nuevos parámetros

En esta sección se introducen dos nuevos parámetros que permitirán mejorar las decisiones locales que tome el Ant System. Estos son ‘min_visib’ y ‘min_prob’.

Parámetro min_visib:

En el momento de elegir una clase para una pareja (s, h) dada en base a la visibilidad, este parámetro permite reducir la cantidad de clases candidatas a aquellas que realmente puedan conducirnos a buenas soluciones. Se busca que las clases candidatas cumplan en buen grado las preferencias locales, relativas a la visibilidad.

Su funcionamiento es el siguiente: Si según la función de visibilidad, la visibilidad de una clase es menor que min_visib, se considera que no es visible, es decir, se le asigna una visibilidad cero. Esto hace que la misma no sea considerada en el sorteo para un salón y horario dados, ya que su valor de probabilidad final será cero.

Parámetro min_prob:

En la etapa de diseño se decidió recorrer las pareja de (s, h) y elegir la clase que se asignaría a cada una. Una desventaja de este diseño es que el algoritmo se comporta en forma “codiciosa” a la hora de elegir una clase para una pareja de salón y horario, ya que siempre buscará asignar una clase, no importa si el conjunto de clases candidatas tiene buenos elementos para la pareja (s, h) considerada o no. Puede suceder que en determinado paso en la recorrida de parejas (s, h) la mejor alternativa no sea elegir entre las clases disponibles, sino directamente elegir la clase “vacía” (o sea no elegir ninguna clase) ya que ninguna de las clases forma con la pareja de salón y horario una terna interesante.

Se debió elegir un criterio para permitirle al algoritmo tomar la decisión de no asignar ninguna clase a una pareja de salón y horario cuando así resultara más beneficioso. Para ello una alternativa sería verificar que ninguno de los valores de probabilidad de las clases candidatas es bueno, comparándolo con un determinado valor de probabilidad considerado bueno: **min_prob**. Este valor podría determinarse empíricamente.

Eventualmente podría decidirse sortear nuevamente, pero aquí la asignación de la clase “vacía” tiene como objetivo permitir una mayor exploración del espacio de soluciones.

De acuerdo al funcionamiento del Ant System, dados un salón y horario, la hormiga construirá una recta de probabilidades para sortear una clase. Si la clase sorteada tenía en la recta una probabilidad menor a min_prob, es decir, no era en realidad una

buena candidata, entonces no se asignará ninguna clase al salón y horario considerados. Este parámetro trata de evitar la “ansiedad” del algoritmo por asignar **siempre** una clase en cada (s, h). De este modo, la hormiga podrá con cierta probabilidad dejar una pareja (s, h) libre y pasar a la siguiente. Este mecanismo equivale al sorteo de la “clase nula”, con la ventaja que es más “inteligente”.

8.2 Mejor hormiga

En las pruebas anteriores se observó que los rastros conducían a soluciones de un costo medio. Para tratar de mejorar este comportamiento, se implementó una variante de Ant System en la cual de todas las hormigas que intervienen en una iteración, se toma la que mejor solución obtuvo, y la matriz de rastros es actualizada únicamente con el rastro de esa hormiga. De este modo sólo aumentarán los “mejores” rastros, y los correspondientes a soluciones mediocres no quedarán registrados.

Algunas de las pruebas realizadas con esta versión son:

prueba	max_iter	cant_horm	exp_rastro	exp_visib	mult_rastro	max_visib	min_rastro	solucion	iteración
f1	6000	5	0.9	0.2	2	32000	1	13.17	3333
f2	6000	5	0.9	0.2	2	32000	1	15.97	1768
f3	2000	10	0.8	0.2	2	5	5	17.01	1177
f4	2000	10	0.5	0.5	2	5	5	15.54	922
f5	2000	10	2.5	0.5	2	5	5	12.64	920
f6	2000	10	2.5	0.5	10	5	5	13.57	583

Los siguientes parámetros se mantuvieron fijos para todas las pruebas:

coef_evap	0.9995
max_costo	400
min_visib	0
min_prob	0.95

Como se ve, se obtienen buenos resultados, y variando los parámetros se logra llegar antes a una solución casi óptima. Debe observarse que dadas las características del juego de datos de prueba, es imposible que el costo de desperdicio de salón sea menor a 10, por lo tanto el valor real de la solución es, en la prueba f6 por ejemplo, 3.57.

Estos resultados son buenos, quizá mejores que los del Ant System “clásico”, pero de todos modos no se alcanza una convergencia al óptimo.

8.3 Asignación en bloque

Una estrategia que mejora considerablemente la asignación es la descrita en el punto 5.2.2.2.4. Mejoras al algoritmo básico, Asignación en bloque. La asignación en bloque busca preservar la “simetría” en la asignación, es decir, que todas las clases de un curso y grupo se asignen al mismo horario y salón a lo largo de la semana. La misma se implementó solamente para las clases teóricas, dada su mayor importancia y

menor número; es decir, si existen varios grupos para un curso, se observa en general que las clases teóricas son compartidas por todos los grupos, mientras que las prácticas son distintas para cada grupo.

La implementación realizada es la siguiente:

a. En seguida de asignar una clase a un salón y horario, se consulta si la misma es teórica. De ser así se pasa a la función encargada de la asignación en bloque.
b. Se consultan las ubicaciones de la clase en cuestión, es decir, a qué carreras, años, cursos y grupos pertenece.
c. Para cada ubicación de la clase, se obtienen las restantes clases teóricas del grupo (dentro de un curso en particular).
d. Para cada clase, si la misma no había sido ya asignada, se obtienen sus rangos horarios de preferencia.
e. Respetando estos rangos horarios y las restricciones que debe cumplir la clase para ser asignada, se cargan en vectores los salones y horarios (días de la semana, ya que el horario será el de la clase original) tentativos para asignar la clase.
f. Se sortea uno de los salones y horarios tentativos y se asigna la clase, quedando así ésta en otro día de la semana, a la misma hora que la clase original.

La implementación tiene en cuenta que podría ser que la asignación en bloque no fuera conveniente. Por ejemplo, si para el primer teórico de un curso se especifica cierta hora del día en particular y para el segundo otra hora distinta, no tiene sentido pretender asignar “en bloque”. Por ello se respetan los rangos horarios de preferencia de cada clase teórica. Si éstos son compatibles, entonces se buscará asignar en bloque.

Otra característica de la implementación es el **sorteo** entre los salones y horarios tentativos. Podría haberse considerado el espaciamiento en los días de la semana de las clases, pero esto conduciría a un algoritmo mucho más complejo, ya que implicaría conocer la cantidad de clases por semana para cada grupo en donde se encontrara la clase original, saber de antemano cuáles de las clases serían pasibles de ser asignadas en bloque según sus rangos horarios de preferencia y el cumplimiento de las restricciones, etc. Por otra parte el espaciamiento es un componente de la función objetivo, y es allí donde será controlado.

9 Pruebas(II) – Juego de datos reales

De la reunión con el Prof. Marcelo Cerminara se recabó información acerca de la realidad del problema de asignación de cursos a salones y horarios restringido a primer año. A partir de lo recabado se confeccionó un juego de datos de prueba, el cual es lo suficientemente complejo y tiene aristas que lo hacen interesante para ser entregado al algoritmo para su procesamiento.

En esta sección veremos primeramente una descripción del juego de datos de prueba reales; dadas las características del mismo, se introduce el concepto de "maxigrupo". En el punto 9.2 se describen las pruebas realizadas, se analizan los resultados y los problemas detectados, aplicando una técnica espiral, en la cual cada etapa consta de nuevas pruebas y evaluación de los resultados. Con el objetivo de acelerar la convergencia a buenas soluciones, en el punto 9.3 se estudian mejoras a las funciones de visibilidad y rastro. Estas se prueban en el punto 9.4. Finalmente se exponen las conclusiones en 9.5.

9.1 Juego de datos de prueba reales

Se considera una sola **carrera** que comprende las materias comunes a ciclo básico y computación, y un solo **año**, primero.

Se consideran los siguientes **cursos**:
- Cálculo 1
- Algebra
- Física Gral. 1
- Taller de Diseño

Se cuenta además con tres **turnos**, mañana, tarde y noche en los cuales de van a asignar las clases. Estos tres turnos tienen la particularidad de no ser disjuntos en lo que se refiere a los rangos horarios que los definen.

Si los turnos fueran disjuntos y dado que es razonable pensar que el problema debería guardar características similares en la asignación de las clases en cada turno, sería altamente probable que la dificultad de lograr una asignación de los tres turnos no fuera mucho mayor que la de asignar uno solo, es decir clases de diferentes turnos no competirían por salones y horarios debido a la restricción de pertenencia a un determinado turno.

Pero como los turnos se superponen y bastante, la competencia entre clases por una pareja de salones y horarios es mayor, dado además que estamos hablando de clases numerosas y que la facultad cuenta con pocos salones de capacidad adecuada para impartir los teóricos. Por lo tanto es importante realizar pruebas considerando la totalidad de los turnos.

9.1.1 Maxigrupos

Surge con este juego de datos el objetivo de repartir a los alumnos en distintos grupos, los cuales abarcarán varios cursos. Podríamos llamar a estos grupos “maxigrupos”, para diferenciarlos de la idea de grupo dentro de un curso. Cada estudiante deberá asistir a uno de estos maxigrupos, y necesitará conocer un esquema de horarios de todos los cursos de su maxigrupo.

Los estudiantes deberán distribuirse en una cantidad de grupos razonable de manera que se pueda asegurar que los mismos asistan a clase dentro de los grupos que tienen asignados y así evitar que ciertos teóricos o prácticos se vean superpoblados en detrimento de otra clases que terminen siendo impartidas a pocos alumnos.

Actualmente no se cuenta más con el salón de actos para dictar clases, y eso significa que se ha removido un recurso importante ya que tenía capacidad para unas 400 personas y servía de comodín a la hora de dar clases a grupos numerosos. Ninguno de los salones de facultad tiene tal capacidad; quizás los que tienen mayor capacidad son el salón 107 para aproximadamente 170 personas y el 301 para 150.

En base a las anteriores consideraciones y a los siguientes parámetros se determina inicialmente la cantidad de grupos de estudiantes en cada curso:

Cantidad de estudiantes que se estima cursarán un año
Cantidad de estudiantes que se espera en cada curso, grupo y clase
Capacidad y otras características de los salones de facultad disponibles
Cantidad de docentes que disponen los institutos para asignar al dictado de las clases correspondientes.

De esta manera se arma el juego de datos iniciales, que consiste en definitiva en un conjunto de “maxigrupos”, cada uno compuesto de un conjunto de clases teóricas y prácticas correspondientes a los cursos de primer año y que serán destinados a un conjunto de determinado de estudiantes. Cada maxigrupo está incluido dentro de un solo turno.

Es evidente que una asignación válida debe evitar la superposición de las distintas clases de ese maxigrupo ya sea en el tiempo (horarios) o en el espacio (salones).

9.1.2 Utilización de los resultados de la asignación

Los resultados de una asignación serán utilizados por un lado por los profesores que son los responsables de impartir las clases, y por los diferentes institutos para que estos puedan tener un registro de cómo está asignado su recurso “docentes” a la tarea de impartir cursos.

Por otro lado los resultados de la asignación son utilizados por la Bedelía de la Facultad y por los estudiantes que se van a inscribir a los cursos de primero.

Es deseable que los resultados de una asignación se muestren de una manera clara que facilite a los estudiantes ver sus alternativas y le facilite en lo posible a la Bedelía la tarea de inscripción y asignación en los diferentes grupos[1].

Con estas consideraciones es claro que la forma más útil de presentar la información de la asignación es por medio de una tabla para cada maxigrupo, donde se muestre para cada día de la semana y horarios dentro de estos días las clases asignadas y los salones correspondientes.

9.1.3 Resumen del juego de datos reales

- una carrera
- un año
- 26 maxigrupos compuestos 4 cursos que son:
- Cálculo 1
  - Clases teóricas
  - Clase practicas
- Algebra
  - Clases teóricas
  - Clase practicas
- Física General
  - Clases teóricas
  - Clase practicas
- Taller de Diseño
  - Clases teóricas
  - Clase practicas
- En total se cuenta con 62 clases
- Turnos Hor. Inicio Hor. Fin
  1 0 12
  2 6 21
  3 16 30
- Cantidad de medias horas en un día: 31
- Cantidad de días en la semana: 5
- Cantidad total de medias horas: 155
- Se cuenta con 19 salones con distintas características.

El juego de datos completo puede verse en el Apéndice A y en el CD-ROM que acompaña al informe.

[1] **Mecanismo de inscripción:**
La bedelía toma la información de cada uno de los grupos o maxigrupos y considera de la misma la cantidad estimada de estudiantes. Al abrirse el período de inscripciones los estudiantes precisan ver las tablas de horarios de cada uno de los grupos de una manera fácil para ayudarlos a determinar sus preferncias. A la hora de inscribirse indican su preferencia a bedelía y ésta intenta asignarlos en los distintos maxigrupos de manera de ir completando los cupos correspondientes, de forma de asegurar de que se llenen de la manera más uniforme posible.

9.1.4 Variantes en las características de los juegos de datos

De acuerdo a las distintas etapas o años de una carrera, surgen distintas situaciones que se hace necesario considerar a la hora de construir el juego de datos para la asignación.

En los primeros dos años por ejemplo, es importante el concepto ya descripto de **"maxigrupo"**, debido al gran número de estudiantes y a la uniformidad de sus necesidades en cuanto a las materias a cursar.

En años más avanzados, el número de estudiantes se reduce, así como la cantidad de docentes por curso, y además cada estudiante cursa un subgrupo de materias que puede abarcar más de un año, y ser muy distinto al de otro estudiante. En este caso, pierde utilidad la idea de "maxigrupo", y pasa a ser útil la idea de **"perfil"**.

La representación del problema con las estructuras de datos elegidas, nos permite representar el concepto de maxigrupo en los juegos de datos en que es necesario[2] y de obviarlo en los juegos de datos en que no, así también como la posibilidad de especificar perfiles.

Los maxigrupos se distinguen asignándole a cada grupo dentro de cada curso el mismo identificador de grupo (ver sección 6.3.3.).

Por todo esto vemos que nuestro algoritmo debe poder ser sometido a juegos de datos de diversas características:

a) juegos de datos orientados a asignar primer y segundo año donde el concepto de maxigrupo puede tener un peso importante.
b) juegos de datos de años posteriores en computación o ciclos técnicos donde el concepto de maxigrupo puede ser irrelevante.
c) juegos de datos completos y complejos donde se quieran asignar cursos de primer año, de otros años y eventualmente todos los cursos de todos los años de todas las carreras de facultad donde en algunos casos se consideraran maxigrupos y en otros no dentro del mismo juego de datos.

[2] **Modificacion de la estructura horscaranio**

Horscaranio se debio adaptar a la realidad de que en el momento de elegirse una clase para asignarse a una pareja sh (asignacion en bloque en realidad) debia no solo comprobarse que no hubiera ninguna clase de algun otro curso dentro del mismo año y carrera ya asignada y de si ya existia una clase del mismo curso asignada asegurarse que fuera de un grupo diferente.

Ahora debe considerarse que a la hora de elegir una clase para asignar a una derterminada pareja de sh (y el restos del bloque) hay que asegurarse que ninguna otra clase del mismo maxigrupo haya sido asignada a la misma.

9.2 Pruebas y resultados

Para hacer las pruebas inicialmente se consideró uno solo de los turnos, de manera de ir comprobando como evolucionan las asignaciones con un grupo mas manejable de datos, y con pruebas que requerirían menos tiempo de procesamiento.

9.2.1 Descripción del subconjunto de datos utilizado

Se eligió el turno de la mañana, que consta de 62 clases y 12 maxigrupos.

Se asignaron capacidades a los salones acordes a la realidad en la Facultad.

Se consideraron diferentes políticas para la determinación de los rangos horarios de preferencia de cada clase; en las primeras pruebas consideramos la asignación efectiva de cursos a salones y horarios encontrada por profesores del Instituto de Matemáticas (asignación del primer semestre del año 98) e hicimos coincidir más o menos ajustadamente los rangos de preferencia con los horarios asignados en la realidad (se utilizaron rangos horarios que incluían a los asignados con un poco de holgura). Esto permitió comprobar si el algoritmo obtenía soluciones parecidas a la real y a la vez ir calibrando los parámetros del problema. Si la solución alcanzada por el IMERL era una solución factible al problema y además contemplaba las preferencias de asignación que hemos relevado, era muy probable que nuestro algoritmo debería converger a una solución muy parecida a aquella en lo que respecta a los horarios asignados a las clases.

Luego en pruebas posteriores se amplió el rango horario de preferencias de la clase de modo que abarcara todo el día o a más de un día. Con esto se pretendía darle más libertad al algoritmo y comprobar qué resultados se obtenían. Además esto estaría más cerca de un caso real, en el cual probablemente se tenga una idea de qué día o días serían preferibles para una clase dada, pero no se pueda determinar exactamente en qué horario sería mejor asignarla.

Más adelante se buscó que el algoritmo trabajara más duro para ir obteniendo soluciones que mejoraran otras características como ser el desperdicio y capacidad del salón, otras características del salón, adyacencia de teóricos y prácticos en los horarios y los salones, compactibilidad de la solución, espaciamiento de las clase de un mismo grupo en los días de la semana, etc.

9.2.2 Ejemplos de pruebas realizadas

La totalidad de los datos de cada prueba puede encontrarse en el cd-rom que acompaña el informe, así como otras pruebas no presentadas en esta sección.

Resumen de pruebas A:

En estas pruebas se utilizaron rangos horarios de preferencia similares a los rangos horarios asignados a cada clase en la asignación realizada por el IMERL.

test	a1	a2	a3	a4	a5
max_iter	10	200	500	500	500
cant_horm	5	10	10	10	10
exp_rastro	0.2	0.2	0.2	0.1	0.5
exp_visib	0.8	0.8	0.8	0.9	0.5
coef_evap	0.995	0.995	0.995	0.995	0.995
peso_ran_hor	0.1	0.1	0.1	0.1	0.1
peso_capac_salon	0.25	0.25	0.25	0.25	0.25
peso_otros_req	0.25	0.25	0.25	0.25	0.25
peso_desper_salon	0.25	0.25	0.25	0.25	0.25
peso_espaciamiento	0.25	0.25	0.25	0.25	0.25
peso_adyacencia	0.3	0.3	0.3	0.3	0.3
peso_compactibilidad	0.2	0.2	0.2	0.2	0.2
max_costo	4000	4000	4000	4000	4000
mult_rastro	50000	25000	25000	25000	25000
min_visib	0.15	0	0	0	0
max_visib	3	3	3	3	3
min_prob	0.96	0.96	0.95	0.95	0.95
min_rastro	20	20	20	20	20
Mejor solución	49.05	56.85	62.3	48.85	63.4
Iteración	4	79	18	1	408

Veremos como ejemplo la prueba A4, y de ella, cómo resultó la asignación para los grupos H01 y H03. Recordar que las pruebas se han efectuado considerando sólo el turno de la mañana, desde la media hora 0 a la 12 (o sea de 8:00 a 14:30).

Prueba A4:

Grupo h01:

1/2 hora	dia 0	dia 1	dia 2	dia 3	dia 4
h 0		cp0020_ma 002	alp002_mi 008	fgp046_ju 001	
h 1	alp002_lu 001	cp0020_ma 002	alp002_mi 008	fgp046_ju 001	cp0020_vi 002
h 2	alp002_lu 001	cp0020_ma 002	alp002_mi 008	fgp046_ju 001	cp0020_vi 002
h 3	alp002_lu 001	fgp046_ma 007			cp0020_vi 002
h 4	ct0003_lu 001	fgp046_ma 007	ct0003_mi 101	ct0003_ju 202	tdp73a_vi 101
h 5	ct0003_lu 001	fgp046_ma 007	ct0003_mi 101	ct0003_ju 202	tdp73a_vi 101
h 6	ct0003_lu 001		ct0003_mi 101	ct0003_ju 202	tdp73a_vi 101
h 7					tdp73a_vi 101
h 8		fgt002_ma 001		fgt002_ju 110	tdp73a_vi 101
h 9		fgt002_ma 001		fgt002_ju 110	tdp73a_vi 101
h10	alt003_lu 001	fgt002_ma 001	alt003_mi 110	fgt002_ju 110	
h11	alt003_lu 001	fgt002_ma 001	alt003_mi 110	fgt002_ju 110	tdt004_ju 001
h12	alt003_lu 001		alt003_mi 110		tdt004_ju 001

Grupo h03:

1/2 hora	dia 0	dia 1	dia 2	dia 3	dia 4
h 0		alp001_ma 007		cp0030_ju 007	
h 1		alp001_ma 007		cp0030_ju 007	
h 2		alp001_ma 007		cp0030_ju 007	alp001_vi 007
h 3					alp001_vi 007
h 4	ct0003_lu 001	cp0030_ma 001	ct0003_mi 101	ct0003_ju 202	alp001_vi 007
h 5	ct0003_lu 001	cp0030_ma 001	ct0003_mi 101	ct0003_ju 202	tdp060_vi 105
h 6	ct0003_lu 001	cp0030_ma 001	ct0003_mi 101	ct0003_ju 202	tdp060_vi 105
h 7	fgp040_lu 007		fgp040_mi 008		tdp060_vi 105
h 8	fgp040_lu 007	fgt002_ma 001	fgp040_mi 008	fgt002_ju 110	tdp060_vi 105
h 9	fgp040_lu 007	fgt002_ma 001	fgp040_mi 008	fgt002_ju 110	tdp060_vi 105
h10	alt003_lu 001	fgt002_ma 001	alt003_mi 110	fgt002_ju 110	tdp060_vi 105
h11	alt003_lu 001	fgt002_ma 001	alt003_mi 110	fgt002_ju 110	tdt004_ju 001
h12	alt003_lu 001		alt003_mi 110		tdt004_ju 001

Resumen de pruebas B

Este conjunto de pruebas es muy similar al conjunto A. Existen diferencias en el coeficiente de evaporación, en mult_rastro y en min_rastro, y además los pesos de las preferencias están calibrados dando más importancia a los rangos horarios de preferencia. Esto último debe tenerse en cuenta al comparar los resultados, ya que los pesos de las preferencias afectarán el valor absoluto de la solución. Los resultados obtenidos son similares a los obtenidos en el conjunto de pruebas A.

test	b1	b2	b3	b4
max_iter	800	200	250	200
cant_horm	5	5	5	5
exp_rastro	0.2	0.2	0.2	0.2
exp_visib	0.8	0.8	0.8	0.8
coef_evap	0.9	0.9	0.995	0.995
peso_ran_hor	0.5	0.5	0.5	0.5
peso_capac_salon	0.3	0.3	0.3	0.3
peso_otros_req	0.17	0.17	0.17	0.17
peso_desper_salon	10	10	10	10
peso_espaciamiento	0.17	0.17	0.17	0.17
peso_adyacencia	0.1	0.1	0.1	0.1
peso_compactibilidad	0.1	0.1	0.1	0.1
max_costo	4000	4000	4000	4000
mult_rastro	500	200000	200000	100000
min_visib	0	0	0.15	0.15
max_visib	3	3	3	3
min_prob	0.95	0.95	0.96	0.96
min_rastro	100	100	50	50
Mejor solución	133.91	84.38	33.08	31.78
Iteración	333	158	9	102

Evaluación del resultado de las pruebas B

Se corrió con no demasiados pasos de iteración para centrarnos en la evolución inicial del algoritmo. Se comprobó que rápidamente el algoritmo encontraba soluciones factibles, y además las soluciones eran aceptables desde el punto de vista de la satisfacción de las preferencias de la asignación.

Se detecta una evolución inicial en la calidad de las soluciones que se iban encontrando: la gráfica de costos versus numero de iteración registra un descenso desde el comienzo que luego se modera con el correr de las iteraciones.

Los salones asignados a las clases cumplen las preferencias aceptablemente, aunque todavía se podría mejorar este aspecto para que se ajustaran más a la cantidad de estudiantes que debían albergar, y evitar un poco más el desperdicio de capacidad en los mismos.

La solución arrojada por el algoritmo respeta en buena medida los rangos de horarios elegidos para la mayoría de las clases, mostrando que la asignación en bloque de las clases de un mismo grupo se cumplía como se esperaba. Algunas clases quedaban fuera de los días en los que se especificaba el rango de horario de preferencias; se observa que en estos casos es determinante la fuerte competencia entre las clases por ciertas parejas de salones y horarios ya que los rangos eran rígidos y no dejaban mucha libertad de elección a una clase para satisfacer sus preferencias en ese aspecto.

Todavía otras preferencias de adyacencia o compactibilidad no alcanzan los mejores valores, pero esto es natural si consideramos que las pruebas hasta este punto se efectuaron con relativamente pocos pasos de iteración.

Problemas detectados en pruebas B

Se estudió la evolución de la matriz de rastros con el avance de las iteraciones. Inicialmente se observó la tendencia de que para una determinada clase, los rastros se acumularan en pocos puntos correspondientes a parejas de salón y horario; es deseable que las hormigas cuenten con capacidad para explorar el espacio de soluciones y no se rijan tempranamente por los rastros correspondientes a soluciones mediocres.

En las primeras corridas se observaba un aumento importante en términos relativos en la cantidad de rastros en los sectores de la matriz que correspondían a ternas de las soluciones encontradas desde el comienzo. Y como esas soluciones no son necesariamente muy buenas, la incidencia de los rastros en la elección de una clase puede influir negativamente en las siguientes hormigas de manera que no puedan tomar las mejores decisiones localmente.

Existe un grupo de parámetros (mult_rastro, coef_evap, min_rastro) muy relacionado con la aplicación de los rastros dejados por las hormigas en su camino que deben ser calibrados entre sí para permitir una adecuada evolución de los rastros, como ya se describió en la sección 7.2.

La evolución de los rastros en la matriz debe ser tal que permita reflejar el descubrimiento de buenos caminos de manera que luego resulten lo suficientemente atractivos como para que otras hormigas los tengan en cuenta a la hora de tomar sus decisiones de locales (pasos) en la construcción de la asignación.

No debe permitirse una acumulación rápida y excesiva de los rastros al encontrar soluciones factibles en las primeras iteraciones del algoritmo, de manera de evitar la convergencia temprana a soluciones de calidad mediocre.

Se debe lograr que la evaporación sea lo suficientemente alta como para contribuir a que la hormiga “se olvide” de las decisiones equivocadas o poco interesantes y solo sobrevivan los caminos que tengan aristas de real interés. Por otro lado, se debe evitar que la evaporación sea demasiado alta de manera que los rastros de esfumen muy pronto, antes de poder influir en la decisión de las hormigas subsiguientes, y de esa forma se pierda el efecto autocatalítico que permite que las siguientes hormigas se basen en los datos aportados por las demás.

Un efecto que podría favorecer una acumulación de rastros para ternas mediocres y no óptimas es el siguiente: Puede ocurrir que ternas que constituirían una solución de muy buena calidad tienen la “mala suerte” de no ser elegidas una cantidad suficiente de veces como para poder aumentar el valor del rastro de manera de contrarrestar el efecto de las evaporaciones sucesivas; de esta forma ternas interesantes podrían tener asociado un rastro mucho menor que alguna otra terna que tuvo la “suerte” de ser elegida en las primeras iteraciones y el rastro que se le aportó contribuyó a que fuera luego elegida con mas frecuencia.

Se probaron distintos valores de min_rastro y mult_rastro y se observó que los mejores comportamientos se dan cuando la función de rastro para una terna de salón, horario y clase toma valores pequeños en relación al valor de min_rastro, de manera que los rastros registren pequeños aumentos y evoluciones suaves.

Resumen de pruebas C

En la prueba C1 se amplía el rango horario de preferencia de cada clase de modo que abarque todo un día (dentro del turno que le corresponde), el día al que pertenecía el rango utilizado anteriormente
En las pruebas C2 a C4 el rango se amplía a 2 días consecutivos. Con esto se busca simular mejor la realidad, en la cual probablemente se tenga una idea de la ubicación de una clase, pero no se pretenda asignarla en un horario definido

Prueba	C1	C2	C3	C4
max_iter	150	300	150	200
cant_horm	10	10	10	10
exp_rastro	0.5	0.2	0.2	0.2
exp_visib	1.7	0.8	0.8	0.8
coef_evap	0.995	0.995	0.995	0.995
peso_ran_hor	0.5	0.1	0.1	0.1
peso_capac_salon	0.5	0.25	0.2	0.2
peso_otros_req	0.5	0.25	0.15	0.15
peso_desper_salon	0	0.25	0.15	0.15
peso_espaciamiento	0.5	0.25	0.15	0.15
peso_adyacencia	0.3	0.2	0.15	0.15
peso_compactibilidad	0.2	0.3	0.7	0.7
max_costo	4000	4000	4000	4000
mult_rastro	200000	25000	25000	50000
min_visib	0.25	0	0	0
max_visib	3	3	3	3
min_prob	0.75	0.95	0.95	0.95
min_rastro		20	20	20
Solución	70.7	50.95	48.35	61.4
Iteración	34	244	12	58

Resultados

En este caso fue interesante comprobar que el algoritmo rápidamente encontraba soluciones factibles y que además la gran mayoría de las clases eran asignadas dentro de los días de su preferencia.

Ejemplo: Prueba C4

Grupo H01

	dia 0	dia 1	dia 2	dia 3	dia 4
h 0	cp0020_ma 107		fgp046_ma 107		tdp73b_vi 107
h 1	cp0020_ma 107		fgp046_ma 107		tdp73b_vi 107
h 2	cp0020_ma 107		fgp046_ma 107		tdp73b_vi 107
h 3		fgt002_ma 008		fgt002_ju 113	tdp73b_vi 107
h 4		fgt002_ma 008		fgt002_ju 113	tdp73b_vi 107
h 5		fgt002_ma 008		fgt002_ju 113	tdp73b_vi 107
h 6	alp002_lu 107	fgt002_ma 008	alt003_mi 101	fgt002_ju 113	cp0020_vi 008
h 7	alp002_lu 107	alt003_lu 008	alt003_mi 101	tdt004_ju 002	cp0020_vi 008
h 8	alp002_lu 107	alt003_lu 008	alt003_mi 101	tdt004_ju 002	cp0020_vi 008
h 9	ct0003_lu 001	alt003_lu 008	ct0003_mi 110		ct0003_ju 007
h10	ct0003_lu 001	fgp046_ju 108	ct0003_mi 110	alp002_mi 001	ct0003_ju 007
h11	ct0003_lu 001	fgp046_ju 108	ct0003_mi 110	alp002_mi 001	ct0003_ju 007
h12		fgp046_ju 108		alp002_mi 001	

Grupo H03

	dia 0	dia 1	dia 2	dia 3	dia 4
h 0				alp001_vi 107	tdp060_vi 008
h 1				alp001_vi 107	tdp060_vi 008
h 2				alp001_vi 107	tdp060_vi 008
h 3	fgp040_lu 107	fgt002_ma 008	alp001_ma 107	fgt002_ju 113	tdp060_vi 008
h 4	fgp040_lu 107	fgt002_ma 008	alp001_ma 107	fgt002_ju 113	tdp060_vi 008
h 5	fgp040_lu 107	fgt002_ma 008	alp001_ma 107	fgt002_ju 113	tdp060_vi 008
h 6	cp0030_ma 108	fgt002_ma 008	alt003_mi 101	fgt002_ju 113	
h 7	cp0030_ma 108	alt003_lu 008	alt003_mi 101	tdt004_ju 002	
h 8	cp0030_ma 108	alt003_lu 008	alt003_mi 101	tdt004_ju 002	
h 9	ct0003_lu 001	alt003_lu 008	ct0003_mi 110	cp0030_ju 008	ct0003_ju 007
h10	ct0003_lu 001	fgp040_mi 105	ct0003_mi 110	cp0030_ju 008	ct0003_ju 007
h11	ct0003_lu 001	fgp040_mi 105	ct0003_mi 110	cp0030_ju 008	ct0003_ju 007
h12		fgp040_mi 105			

Se estudia la matriz de rastros resultante y se constata que dada una clase, los mejores rastros se producen en el entorno de los horarios que pertenecen a la franja de preferidos en su vector de preferencias. Se dan valores interesantes de rastro en más de una pareja (s, h), lo que es un buen indicador de que la hormiga tiene capacidad de exploración y no se queda con una sola solución sino que investiga otras alternativas en el espacio de soluciones. Se observa que los rastros mantienen cierta uniformidad entre sí no registrándose grandes variaciones entre sus valores, hecho que es importante si consideramos que estamos corriendo con no demasiadas iteraciones y las solución final todavía no es óptima.

En pruebas subsiguientes se prueba con distintos valores de min_prob disminuyendo el valor de este parámetro con el objetivo de lograr que el algoritmo sortee la clase vacía con mayor frecuencia en el caso de que la clase sorteada no esté entre las mejores candidatas en términos relativos.

9.3 Ajuste de parámetros y mejoras al algoritmo

9.3.1 Ajuste de parámetros

Observaciones sobre el funcionamiento de min_prob

La función de probabilidad es calculada para cada clase que cumple con las restricciones, luego el valor de probabilidad para todas las clases es normalizado según el mayor de los valores registrados entre las clases de manera que los valores obtenidos pertenezcan al intervalo [0,1] donde 1 corresponde a la clase que mayor valor de probabilidad había obtenido entre las candidatas. Esta normalización es la que hace posible el uso de **min_prob**.

El filtro “prob(clase) < min_prob” significa que la clase en cuestión sólo será admitida si tiene un valor de prob que es mayor al min_prob*100 por ciento del valor de prob de la clase mejor calificada.

¿Pero qué ocurre si no existe entre las candidatas una clase que sea realmente una buena opción? Si todas las clases candidatas fueran “mediocres”, y todas con probabilidades similares, al normalizar, todas tendrían probabilidades cercanas a 1, y min_prob no sería suficiente para determinar que en ese caso sería beneficioso no sortear ninguna clase, ya que cualquiera de ellas tendría una probabilidad mayor a min_prob.

Sería necesario entonces algún otro criterio que permitiera decidir si la clase no es realmente buena. Por ejemplo, podríamos basarnos en su rastro. Si su rastro fuera alto, entonces sería realmente una buena candidata. De lo contrario sería conveniente elegir la clase nula.

9.3.2 Mejoras al algoritmo

9.3.2.1 Mejora a la función de visibilidad

Inicialmente la función de visibilidad se definía de la siguiente forma:

$$\text{Visib} = \begin{cases} \text{Max\_visib} & \text{si costo\_vis} = 0 \\ 1/\text{costo\_vis} & \text{si no} \end{cases}$$

Siendo costo_vis el costo de asignar la clase considerada a la pareja (s, h), calculado como se indica en la sección 6.5.2.

En esta definición, si bien podemos asegurar que el recorrido de la función esté comprendido entre 0 y max_visib, es difícil determinar segmentos del recorrido que representen a las distintas calidades que puede exhibir la visibilidad de una clase.
La función de visibilidad para valores de costo por visibilidad mayores que cero corresponde al inverso del costo por visibilidad lo que resulta en un valor comprendido entre 0 y 1. La distribución de estos costos puede variar enormemente en función del

rango real de valores de los costos para cada instancia particular del problema (según distintos juegos de datos).

Es deseable que se registren valores altos de visibilidad en caso de costos bajos, es decir cuando la clase cumple preferencias locales y satisface en buena manera las globales y que por el contrario tengamos valores bajos cuando las clases violan esas preferencias.

Con esta definición se depende fuertemente de la instancia del problema en cuanto a en qué rango de valores se den los costos por visibilidad y por lo tanto se hace difícil determinar segmentos del recorrido de la función de visibilidad que correspondan a buenas, medias y malas visibilidades, así como también es difícil la elección del valor de la constante min_visib, que determina cuándo una clase no debe ser considerada.

Otro problema es la determinación del valor de max_visib. Si max_visib es mayor que uno existe una diferencia importante entre el valor de visibilidad para un costo óptimo y para un costo muy bueno, es decir, la función experimenta una discontinuidad. Esto puede considerarse positivo si se desea asegurar una ubicación óptima para una clase, pero por otro lado, no sólo debe respetarse la opción óptima localmente si no que deben considerarse elecciones no tan buenas localmente pero que desemboquen en una solución global buena.

Se intenta entonces buscar otras funciones que permitan una adaptación más dinámica a la instancia del problema y que muestren una distribución mas homogénea de los valores del recorrido. A continuación se describen algunas soluciones a los problemas expuestos.

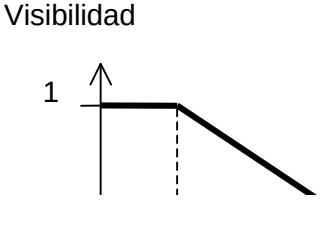
Función de visibilidad dada por una recta por dos puntos

Se busca construir una función de visibilidad tenga las siguientes características:
a) sea una recta de pendiente negativa que pase por los siguientes puntos:

- (CMIN,1) donde CMIN es un valor de costo por visibilidad lo suficientemente bajo como para corresponder a una clase muy visible, que satisface muy bien las preferencias.
- (CMAX, VMIN) donde CMAX es una cota superior al costo por visibilidad y VMIN un valor muy pequeño cercano a cero.

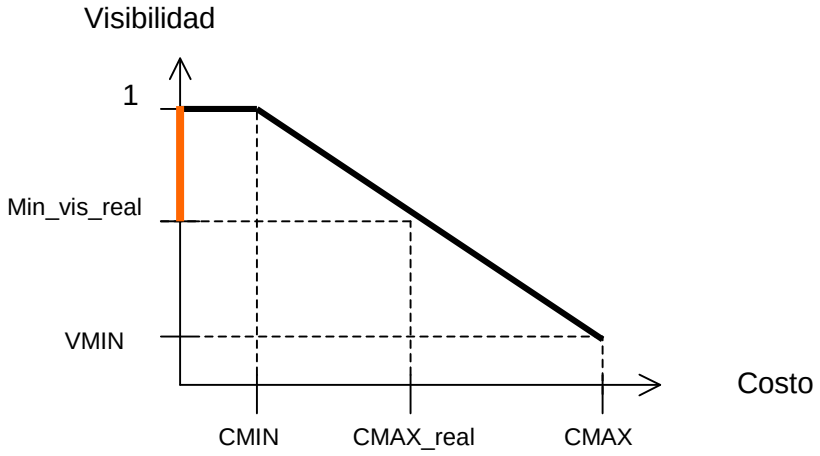
b) Para valores de costo menores que CMIN, la visibilidad será 1.

c) para valores de costo mayores a CMAX la función de visibilidad valdrá 0.



Se busca que la función de visibilidad tome valores entre 0 y 1 para poder controlar los valores en los que puede oscilar. La importancia de poder acotar los valores de la función de visibilidad a un segmento independientemente de la instancia del problema cumple más de un objetivo: la función de probabilidad se obtiene de hacer un producto de visibilidad elevado a un exponente por rastro elevado a otro exponente. Ambos exponentes son parámetros del algoritmo que deben ser calibrados, y que junto a los valores efectivos que tomen la visibilidad y el rastro van a determinar cómo inciden ambos factores en la calificación de las clases candidatas. La tarea de determinación de los exponentes puede ser realizada con mayor facilidad si podemos gobernar el comportamiento de las bases para toda instancia de problema. Si no podemos determinar en qué rango tomarán valores las bases es muy difícil que podamos elegir valores para los exponentes que sean adecuados para todas las instancias del problema y conduzcan a adecuadas evaluaciones relativas entre las clases candidatas.

Se tiene la dificultad de que la cota superior al costo por visibilidad sea un poco gruesa es decir, que sea mucho mayor que el máximo costo efectivo. Eso se traduciría en un aumento en los valores de la función de visibilidad para costos grandes, o sea que se tendrían valores altos de visibilidad para clases que en realidad no serían buenas candidatas. Los valores de visibilidad estarán entre Min_vis_real y 1, y no entre
VMIN y 1:



Una forma de suavizar este problema es minimizar este efecto elevando al cuadrado el valor de visibilidad obtenido; de esta manera se logra distribuir más uniformemente los valores en el intervalo [VMIN, 1] (por ej. 0.3 elevado al cuadrado vale 0.09).

Cómo entonces determinar CMAX ?
La alternativa de fijar un valor estáticamente que se use en todo el desarrollo del algoritmo tendría el riesgo de ser una aproximación gruesa.

Podría ejecutarse una prueba con pocas iteraciones e ir monitoreando los valores que toma el costo por visibilidad y luego estimar un valor para CMAX, el cual se utilizará para la ejecución definitiva del algoritmo. De todas formas se corre el riesgo de tener aproximaciones un tanto gruesas, ya que los valores máximos de costo pueden variar a lo largo de las iteraciones del algoritmo; el valor de costo máximo depende entre otras cosas de cuántas clases queden por asignar y de la disponibilidad de clases candidatas interesantes.

Las pruebas realizadas con esta nueva función de visibilidad se presentan el punto 9.4. (pruebas D4 y D5).

9.3.2.2 Resultados

Se observa que se encuentran rápidamente soluciones factibles que van decreciendo en costo hasta un punto en que dejan de decrecer.

Se estudian los componentes del costo y se observa que el costo por rangos horarios es el determinante en el costo total, ya que no baja de determinado valor.

Se estudia la solución arrojada para observar donde se produce ese costo y se observa que unas pocas clases de práctico se ven desfasadas un día respecto a su rango de preferencias mientras que la mayoría son asignadas dentro del día que corresponde a su rango horario deseado.

Observando cómo se va construyendo la solución, se concluye que el problema es el siguiente: En el momento en que van a elegirse las últimas clases para algunas parejas de salón horario, se cumple que se violaría la preferencia de rango horario, pero como se cumplen algunas otras preferencias finalmente el costo total por visibilidad toma valores aceptables, pero la recta evalúa en valores muy altos para ese valor de costo.

Entonces si bien la recta permite homogeneizar los valores de visibilidad, no es efectiva para penalizar ciertos costos por visibilidad no muy altos que sin embargo corresponden a ternas que violan alguna preferencia importante.

Con la función de visibilidad considerada, una forma de evitar esto sería utilizar un valor alto en min_visib, pero esto haría que se descartara a la mayoría de las clases, lo que dificultaría llegar a soluciones factibles.

Se busca entonces obtener mejor función para calcular la visibilidad:

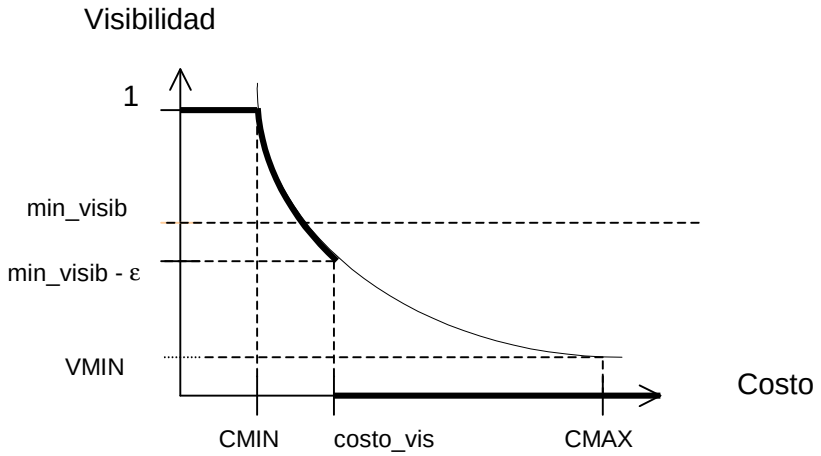
- que sea independiente de los datos del problema, o que sea adaptativa.
- que permita segmentar su recorrido, es decir, para distintos rangos de costo, la función devolverá visibilidades diferentes.
- que premie asignaciones que no implican violación importante de preferencias locales, y que castigue cuando éstas se producen.

9.3.2.3 Nueva función de visibilidad: parábola por tres puntos

Una alternativa es considerar una parábola de pendiente negativa, la cual por su concavidad permite acentuar las diferencias entre visibilidades buenas y malas. Esta parábola deberá pasar por dos puntos, al igual que la recta considerada anteriormente.
Pero por dos puntos pasan infinitas parábolas. Faltaría determinar un tercer punto por donde pasaría esta parábola.

Cómo determinar una parábola que además tenga buenas propiedades de filtrado de clases que no sean buenas candidatas?

Primeramente reemplazamos la antigua recta por una parábola. Gráficamente tenemos:



Se establece un criterio que decida cuando una clase debe tener visibilidad nula: Este consiste en evitar que ninguna clase se asigne en un día que en el cual su rango de preferencias horarias no tiene ninguna porción incluida en él, por lo tanto la idea es que las asignaciones que tengan un costo que indica que la clase no será asignada en su día de preferencia, sean descartadas.

El criterio puede expresarse así:

La función de visibilidad evaluada en cualquier costo por visibilidad que implique que la clase en cuestión será asignada en un horario que no pertenece a ninguno de los días que incluyen porciones de los rangos horarios de preferencia debe ser filtrada por min_visib .

Recordar que para determinar la función de visibilidad se evalúa primero la función de base, sea recta o parábola, en el punto y luego se compara con el valor de min_visib.

Para aquellos valores de costo donde la función de base evalúa en un valor menor que min_visib la función de visibilidad se hará cero y en los otros valdrá lo que evalúe la función de base.

Sea V* la función de base para el cálculo de la visibilidad y llámese 'costo_vis' a la distancia en medias horas de ocurrencia entre dos turnos multiplicado por el ponderador de costo por desfasaje horario correspondiente.

Demostraremos que una condición necesaria para que se satisfaga el criterio es V(costo_vis) sea menor que min_visib, o sea igual a min_visib - ε ($\varepsilon \cong 0$). Se cumplirá entonces que para valores mayores que costo_vis_rh, la visibilidad será cero. Debemos determinar 'costo_vis' para tener la parábola que será parte de la función de visibilidad totalmente definida.

Es decir que :
Visib(costo_vis) = min_visib - ε

Demostración

Obs:

1) *El costo por visibilidad es siempre mayor o igual que el costo por rh. En el mejor de los casos es costo rh, cuando no haya costo por otros conceptos.*
2) *Dados dos costos por visibilidad c1 y c2 se cumple que*
   si c1 $\leq$ c2 $\Rightarrow$ Visib(c1) $\geq$ Visib(c2).
   La visibilidad es decreciente con respecto al costo por visibilidad.

Queremos probar que:

Visib(costo total correspondiente a 1 o + días de desfasaje) < min_visib

Sólo si

Visib (distancia en media horas entre dos ocurrencias de un turno en días consecutivos * ponderador de costo_rh) = min_visib - ε.

Pero de acuerdo a las observaciones 1) y 2) se cumple que :

Visib (costo total correspondiente a 1 o + días de desfasaje)
< Visib (costo total con un día de desfasaje manteniendo el valor de los otros costos)
< Visib (costo rh con un día de desfasaje)
< Visib (mínimo (costo rh con un día de diferencia))
= Visib (distancia en media horas entre dos ocurrencias de un turno en días consecutivos * ponderador costo_rh)
= Visib (costo_vis)

Por lo tanto si Visib (costo_vis) = min_visib - ε puedo asegurar que las clases que se desfasen uno o más días de su rango horario de preferencia serán descartadas.

Obs.: La menor distancia entre la hora en que el algoritmo asignaría una clase fuera de su día de preferencia y el rango horario de preferencia se da cuando el algoritmo pretende poner la clase al final del turno de uno de los días y el rango de preferencia comienza al inicio del mismo turno el día siguiente, o cuando el algoritmo pretende

poner la clase al inicio del turno y el rango de preferencias termina al final del turno el día anterior.

En el juego de datos reales esta distancia vale 19 medias horas. Entonces el otro punto de la parábola es:

(19 * ponderador_costo_rh, min_visib - ε)

Para asegurar que la condición sea suficiente para satisfacer el criterio expuesto debemos exigir que se cumplan ciertas condiciones. El criterio pretende que cuando una clase se vea “desfasada en menos de un día“ de su rango de preferencia en la asignación tentativa no sea descartada para el sorteo. Si además para esa clase y la pareja de salón y horario correspondiente no se satisfacen otras preferencias, queremos asegurar que la clase no sea descartada. Para que esto se cumpla alcanzará con que se verifique una condición que dependerá de las características de los turnos, de los ponderadores de los costos por visibilidad y de los máximos de las funciones de costo por visibilidad:

Max(costo_vis_por_capacidad_salon) * ponderador_cap_salon +
Max(costo_vis_por_desperdicio_salon)* ponderador_desp_salon +
Max(costo_vis_por_otros_req) * ponderador_otros_req +
Max(costo_vis_por_rh) * ponderador_rh <

distancia en medias horas entre 2 ocurrencias de un turno en días consecutivos * ponderador_rh.

Los valores de costo por r.h. se pueden acotar la longitud del turno. Los valores de costo por otros req. se pueden acotar por la longitud del vector de booleanos con que se implementan. Los costos por capacidad de salón y por desperdicio de capacidad toman valores 0 o 1.

Luego conociendo las características de los turnos sólo resta determinar los valores relativos de los ponderadores para que la condición se satisfaga, tarea que es muy simple y es independiente del juego de datos del problema.

Por lo tanto es factible buscar una parábola decreciente que pase por los puntos
a) (CMIN, 1)
b) (CMAX, VMIN)
c) (19 * ponderador_costo_rh, min_visib - ε)

Una parábola de estas características es de la forma

$F(x)= a/x^2 + b/x +c$ donde x es la variable correspondiente al costo por visibilidad.

Los problemas que quedan por resolver son:

a) Cómo determinar CMAX.
b) Es posible determinar a, b y c algorítmicamente en todos los casos?

a) Trabajaremos con un CMAX adaptativo, éste será recalculado a medida que sea necesario durante la ejecución del algoritmo.
   Para ello:
   Se inicializa CMAX con un valor estimado para su uso en las primeras iteraciones. Se va acumulando el máximo valor de costo por visibilidad en una variable max_cvis. Cada cierto tiempo se actualiza CMAX con el valor max_cvis. Si en determinado instante el valor de CMAX es mucho mayor que el máximo efectivo se debe reducir a max_cvis. Si en cambio en determinado instante el valor de CMAX es menor que el max_cvis debemos aumentar la cota.

b) Veremos un algoritmo que nos permite determinar a , b , c en todos los casos. Ver apéndice B.

9.3.2.4 Mejora a la función de rastros

La función de rastro debe ser decreciente con el costo de la asignación. De modo que basándonos en el TSP, cuya función de rastro es

rastro = K / largo del camino

donde K es una constante que permite regular la cantidad absoluta de rastro, tomamos inicialmente para el problema de asignación la función

rastro = Mult_rastro / costo asignación

donde Mult_rastro hace las veces de la constante K.

Luego se modificó la misma por

rastro = Mult_rastro / costo asignación^2

buscando incrementar las diferencias entre rastros correspondientes a buenas y malas asignaciones, como ya se estudió en el punto 6.8.5.

Utilizando una hipérbola (rastro = K/costo), tenemos una menor diferenciación entre rastros buenos y malos, pero al tener ésta una pendiente menor, es mayor el rango de costos que generan rastros altos. Utilizando la parábola (rastro = K/costo ^{2}) se asignan rastros muy altos a soluciones muy buenas, pero este valor decrece rápidamente con un pequeño aumento en el costo de la solución.

Podríamos aplicar a la función de rastro las mismas ideas que se aplicaron a la función de visibilidad, definiéndola por medio de ciertos puntos, y logrando así un control mucho mayor sobre los valores de rastros que se obtendrán.

Por ejemplo podríamos tomar una hipérbola de la forma A/costo + B que pase por dos puntos: Uno sería (COSTO_MAX, RMIN), donde COSTO_MAX sería una cota

superior al costo de una asignación, y RMIN un valor de rastro cercano a cero. El otro punto sería (COSTO_MIN, RMAX), siendo COSTO_MIN un valor de costo considerado excelente.

Estos valores podrían determinarse de acuerdo a como sería deseable que crecieran los rastros, y de acuerdo a los parámetros min_rastro y coef_evap.

También debería continuizarse la función de rastro en el intervalo [0, COSTO_MIN) haciéndola valer RMAX.

La prueba D6, presentada en el punto 9.4, se realizó utilizando una parábola por 3 puntos como funciones de visibilidad y de rastros.

9.4 Pruebas utilizando las mejoras efectuadas

La siguiente tabla de pruebas muestra las diferencias entre las soluciones obtenidas utilizando distintas funciones de visibilidad y rastros. Los rangos horarios considerados para cada clase abarcan un día. En las pruebas D4 a D6, ya no se utiliza el parámetro mult_rastro, debido a que se utiliza la función de rastro mejorada, y sus valores se controlan automáticamente. Tampoco se utiliza el parámetro max_visib, dado que no es necesario al utilizar la función de visibilidad mejorada.

- Pruebas D1 a D3: utilizan funciones de visibilidad y rastros clásicas (previas a las mejoras).
- Pruebas D4 y D5: utilizan funciones de rastro y visibilidad dadas por una recta por dos puntos.
- Prueba D6: utiliza funciones de visibilidad y rastro dadas por una parábola por 3 puntos.

Resumen de pruebas D

test	D1	D2	D3	D4	D5	D6
max_iter	200	300	400	150	150	100
cant_horm	10	10	10	10	10	10
exp_rastro	0.5	0.5	0.5	0.5	0.5	0.2
exp_visib	1.6	1.6	1.7	1.7	1.7	0.8
coef_evap	0.995	0.995	0.995	0.995	0.995	0.999
peso_ran_hor	0.5	0.5	0.5	0.5	0.5	1.5
peso_capac_salon	0.5	0.5	0.5	0.5	0.5	0.5
peso_otros_req	0.5	0.5	0.5	0.5	0.5	0.5
peso_desper_salon	0	0	0	0	0	0
peso_espaciamiento	0.5	0.5	0.5	0.5	0.5	0.5
peso_adyacencia	0.3	0.3	0.3	0.3	0.3	0.3
peso_compactibilidad	0.2	0.2	0.2	0.2	0.2	0.2
max_costo	4000	4000	4000	4000	4000	4000
mult_rastro	200000	200000	200000	--	--	--
min_visib	0.15	0.15	0.2	0.25	0.15	0.2
max_visib	1	1	1	--	--	--
min_prob	0.96	0.75	0.75	0.75	0.75	0.75
min_rastro	20	50	20	20	20	20
Mejor solución	84.8	154.4	91.6	70.7	88.7	28.6
Iteración	21	220	111	34	14	81

Pruebas D4 y D5

En estas pruebas se utilizó la recta por dos puntos como función de visibilidad, y se estimó un valor de CMAX (costo por visibilidad máximo) que se mantuvo fijo. El valor de visibilidad obtenido se elevó al cuadrado para asegurar una mejor distribución del mismo en el segmento [0, 1].

Los rangos horarios se mantienen correspondiéndose con un día de la semana.

Prueba D4

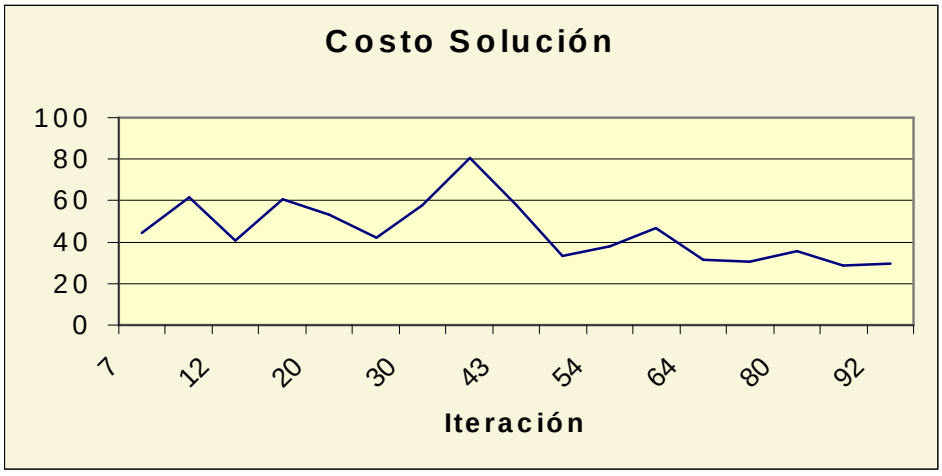
Grupo H01

1/2 hora	dia0	dia1	dia2	dia3	dia4
h 0	ct0003_lu 012	cp0020_ma 000	ct0003_mi 000	ct0003_ju 002	cp0020_vi 000
h 1	ct0003_lu 012	cp0020_ma 000	ct0003_mi 000	ct0003_ju 002	cp0020_vi 000
h 2	ct0003_lu 012	cp0020_ma 000	ct0003_mi 000	ct0003_ju 002	cp0020_vi 000
h 3	alt003_lu 001	fgt002_ma 013	alp002_mi 002	fgt002_ju 000	tdp73a_vi 001
h 4	alt003_lu 001	fgt002_ma 013	alp002_mi 002	fgt002_ju 000	tdp73a_vi 001
h 5	alt003_lu 001	fgt002_ma 013	alp002_mi 002	fgt002_ju 000	tdp73a_vi 001
h 6		fgt002_ma 013		fgt002_ju 000	tdp73a_vi 001
h 7	tdt004_ju 002		fgp046_ma 003		tdp73a_vi 001
h 8	tdt004_ju 002		fgp046_ma 003		tdp73a_vi 001
h 9	alp002_lu 002		fgp046_ma 003		
h10	alp002_lu 002		alt003_mi 003	fgp046_ju 000	
h11	alp002_lu 002		alt003_mi 003	fgp046_ju 000	
h12			alt003_mi 003	fgp046_ju 000	

Grupo H03

	dia0	dia1	dia2	dia3	dia4
h 0	ct0003_lu 012	alp001_ma 001	ct0003_mi 000	ct0003_ju 002	
h 1	ct0003_lu 012	alp001_ma 001	ct0003_mi 000	ct0003_ju 002	
h 2	ct0003_lu 012	alp001_ma 001	ct0003_mi 000	ct0003_ju 002	
h 3	alt003_lu 001	fgt002_ma 013	fgp040_mi 001	fgt002_ju 000	alp001_vi 000
h 4	alt003_lu 001	fgt002_ma 013	fgp040_mi 001	fgt002_ju 000	alp001_vi 000
h 5	alt003_lu 001	fgt002_ma 013	fgp040_mi 001	fgt002_ju 000	alp001_vi 000
h 6		fgt002_ma 013		fgt002_ju 000	tdp060_vi 004
h 7	tdt004_ju 002	cp0030_ma 004		cp0030_ju 000	tdp060_vi 004
h 8	tdt004_ju 002	cp0030_ma 004		cp0030_ju 000	tdp060_vi 004
h 9	fgp040_lu 001	cp0030_ma 004		cp0030_ju 000	tdp060_vi 004
h10	fgp040_lu 001		alt003_mi 003		tdp060_vi 004
h11	fgp040_lu 001		alt003_mi 003		tdp060_vi 004
h12			alt003_mi 003		

Prueba D6



Asignación del Grupo h01

	dia 0	dia 1	dia 2	dia 3	dia 4
h 0	alt003_lu 108	cp0020_ma 008	alt003_mi 001	ct0003_ju 001	tdp73a_vi 007
h 1	alt003_lu 108	cp0020_ma 008	alt003_mi 001	ct0003_ju 001	tdp73a_vi 007
h 2	alt003_lu 108	cp0020_ma 008	alt003_mi 001	ct0003_ju 001	tdp73a_vi 007
h 3	ct0003_lu 002	fgt002_ma 001	alp002_mi 001	fgt002_ju 108	tdp73a_vi 007
h 4	ct0003_lu 002	fgt002_ma 001	alp002_mi 001	fgt002_ju 108	tdp73a_vi 007
h 5	ct0003_lu 002	fgt002_ma 001	alp002_mi 001	fgt002_ju 108	tdp73a_vi 007
h 6		fgt002_ma 001	ct0003_mi 002	fgt002_ju 108	cp0020_vi 007
h 7	tdt004_ju 001		ct0003_mi 002		cp0020_vi 007
h 8	tdt004_ju 001		ct0003_mi 002		cp0020_vi 007
h 9	alp002_lu 002			fgp046_ju 001	
h10	alp002_lu 002	fgp046_ma 101		fgp046_ju 001	
h11	alp002_lu 002	fgp046_ma 101		fgp046_ju 001	
h12		fgp046_ma 101			

Asignación del Grupo h03

	dia 0	dia 1	dia 2	dia 3	dia 4
h 0	alt003_lu 108	cp0030_ma 007	alt003_mi 001	ct0003_ju 001	tdp060_vi 002
h 1	alt003_lu 108	cp0030_ma 007	alt003_mi 001	ct0003_ju 001	tdp060_vi 002
h 2	alt003_lu 108	cp0030_ma 007	alt003_mi 001	ct0003_ju 001	tdp060_vi 002
h 3	ct0003_lu 002	fgt002_ma 001	fgp040_mi 007	fgt002_ju 108	tdp060_vi 002
h 4	ct0003_lu 002	fgt002_ma 001	fgp040_mi 007	fgt002_ju 108	tdp060_vi 002
h 5	ct0003_lu 002	fgt002_ma 001	fgp040_mi 007	fgt002_ju 108	tdp060_vi 002
h 6		fgt002_ma 001	ct0003_mi 002	fgt002_ju 108	
h 7	tdt004_ju 001	alp001_ma 101	ct0003_mi 002	cp0030_ju 007	
h 8	tdt004_ju 001	alp001_ma 101	ct0003_mi 002	cp0030_ju 007	
h 9	fgp040_lu 001	alp001_ma 101		cp0030_ju 007	alp001_vi 008
h10	fgp040_lu 001				alp001_vi 008
h11	fgp040_lu 001				alp001_vi 008

9.5 Conclusiones extraídas del estudio de las pruebas con los juegos de datos reales

Enfrentado a un juego de datos más complejo, el algoritmo igualmente logra obtener soluciones factibles rápidamente, y de calidad aceptable.

Es importante para llegar a buenas soluciones que la matriz de rastros registre acumulaciones en las ternas correspondientes a buenas soluciones; en el caso que existan varias parejas (s, h) interesantes para una clase la matriz debe reflejarlo en su contenido. Esto indica la capacidad de exploración de las hormigas en el espacio de soluciones, evitándose una rápida convergencia a soluciones mediocres. Se implementaron estrategias para lograr un mejor control de la evolución de los rastros, en base al estudio en conjunto de los parámetros min_rastro, coeficiente de evaporación, mult_rastro y de la función de rastro.

El estudio y utilización de los parámetros min_visib y min_prob colaboró para continuar logrando mejoras en las soluciones, y evitar la ansiedad del algoritmo.

La utilización de la variante del algoritmo “mejor hormiga” proporcionó buenos resultados y abre una línea de investigación futura. El número de hormigas utilizado incide más directamente en la calidad de las soluciones obtenidas.

La ‘asignación en bloque’ permitió obtener mejores asignaciones para las clases de un grupo o curso en su conjunto.

Con las modificaciones a la función visibilidad y la función de rastro se mejoraron los resultados. La nueva función de rastro permite impactar más fielmente las características de las buenas soluciones. Estas modificaciones permiten independizarnos en cierta medida del juego de datos de entrada, cosa que antes no era posible.

La función de visibilidad permite por un lado implementar la estrategia ‘greedy’ a través de la cual el algoritmo encuentra soluciones factibles en las primeras iteraciones. Luego, una adecuada función de visibilidad colabora junto con los rastros en la determinación de buenas soluciones en etapas más avanzadas del proceso.

Se observa que la asignación de cada maxigrupo es satisfactoria; sin embargo contiene varios ‘puentes’. Aumentando el número de iteraciones se obtienen mejores soluciones y es de esperar que se obtenga una solución más compacta.

10 Conclusiones finales y trabajo futuro

Conclusiones finales

En el presente trabajo, se comenzó por efectuar un relevamiento del problema de asignación en la Facultad de Ingeniería.

Se realizó una formalización del mismo como un problema de optimización combinatoria, tratando de no perder generalidad en cuanto al alcance del problema.

Utilizando la heurística Ant Systems, se diseñó una solución, la cual fue luego implementada utilizando el lenguaje de programación C++; se implementó además una variante de la misma que llamamos “Mejor Hormiga”.

Se realizaron exhaustivas pruebas, y se estudiaron los resultados y se implementaron mejoras y ajustes de los parámetros del algoritmo, utilizando un proceso de desarrollo en espiral.

No se implementó otra solución utilizando otras heurísticas de las estudiadas; se prefirió en cambio profundizar en la aplicación del Ant System, utilizando un juego de datos de prueba reales.

Se buscó utilizar una representación de la solución análoga a la utilizada en la asignación manual.

En base a la técnica utilizada, el diseño y la implementación particulares, se llegó a una convergencia a un conjunto de buenas soluciones. Un problema inherente al diseño implementado es lograr un compromiso entre una buena asignación, compacta, y evitar la ansiedad del algoritmo.

Trabajo futuro

Todavía se podría continuar estudiando las relaciones entre los parámetros y realizar nuevos ajustes o variantes para acelerar la convergencia y mejorar la calidad de las soluciones. Las funciones de visibilidad y rastro podrían ser objeto de mejoras de manera de aportar mayor inteligencia al algoritmo.

Un criterio alternativo al construir una solución sería no se exigir que la hormiga asigne la totalidad de las clases en una sola recorrida para que esta constituya una solución factible. Una técnica que podría experimentarse es recorrer más de una vez la “lista sh” mientras queden clases pendientes de asignar con el objetivo de llenar los puentes.

Otro punto a considerar para hacer que el algoritmo sirva para automatizar la tarea de asignación sería por un lado, buscar su integración con sistemas de información y complementariamente el desarrollo de interfases adecuadas, ya sea para la entrada de datos como para la mejor visualización de la salida.

11 Cronograma

Tarea a realizar	Período planificado	Cantidad de horas planificada	Período real	Cantidad de horas incurrida
Búsqueda de material relativo al problema y estudio	Abril/mayo	60	Abril/mayo	50
Relevamiento y planteo del problema.		60		40
Formalización del problema	Junio	30	Junio	51
Estudio de Ant Systems y otras metodologías		90		40
Diseño utilizando Ant Systems	Julio	60	Julio	24
Elaboración del primer informe	Julio/agosto	120	Julio/agosto	99
Implementación utilizando Ant Systems	Set/oct	180	Set/oct	250
Pruebas con datos ficticios	Noviembre	120	Noviembre	195
Pruebas con datos reales	diciembre	120	11 diciembre al 30 enero	150
Redacción del Informe final	enero	90	15 enero al 10 marzo	210
Preparación de presentación final	1 al 18 de febrero	60	27 febrero al 20 marzo	100
Total de horas-hombre		990		1209

12 Bibliografía

[1] The Ant System: Optimization by a colony of cooperating agents.
Dorigo,Maniezzo y Colorni.
IEEE Transactions on systems , Man and Cybernetics , Vol.25 , no.12

[2] Distributed Optimization by Ant Colonies. Colorni , Dorigo y Maniezzo. *Appeared in Proceedings of ECAL 91 .Elsevier Publishing , 134-142.*

[3] Ant System: An Autocatalytic Optimizing Process.
Dorigo, Maniezzo y Colorni.
Technical Report 91-016.

[4] Ant-Q. A Reinforcement Learning Approach to Combinatorial Optimization.
Dorigo, Luca María Gambardella.
Technical Report 95-01 IRIDIA Université de Bruxelles

[5] The Ant System Applied to the Quadratic Assignment Problem.
Maniezzo , Colorni , Dorigo
Technical Report 94/28 IRIDIA Université Libre de Bruxelles.

[6] Heuristics from Nature for Hard Combinatorial Optimization Problems
Colorni, Dorigo, Maffioli, Maniezzo, Righini, Trubian.
1996 IFORS. Published by Elsevier Science Ltd.

[7] An Object Oriented Methodology for Solving Assignment Type Problems with Neighborhood Search Techniques.
Ferland, Hertz, Lavoie.
Operations Research, Marzo-Abril 1996.

[8] The Combinatorics of Timetabling
D. de Werra
EJOR 96 (1997) 504-513

[9] Distribution Requirements and Compactness Constraints in School Timetabling.
Andreas Drexl, Frank Salewski.
EJOR 102 (1997) 193-214.

[10] Genetic Algorithms and Highly Constrained Problems: The Time-Table Case.
Colorni, Dorigo, Maniezzo.
Publicado en:: Proceedings of the First International Workshop on Parallel Problem Solving from Nature.

[11] Genetic Algorithms in Solving Constraint Satisfaction Problems: The timetable Case.
Witold Salwach.
Institute of Industrial Engineering and Management, Technical University of Wroclaw, Poland.

[12] Tabu Search Techniques for Large High-School Timetabling Problems.
A. Schaerf
Centrum voor Wiskunfe en Informatica Cs-R9611 1996

[13] Neural Nets
K. Gurney.
http: //www2.shef.ac.uk/psychology/gurney/hotes

[14] Tabu Search for Large Scale Timetabling Problems
A. Hertz
European Journal of Operational Research 54 (1991) 39-47

[15] A Tabu Search algorithm for computing an operational timetable
Daniel Costa
European Journal of Operational Research 76 (1994) 98-110

[16] Introducción y Tutorial de GAMS,
A. Brook, D. Kendrick, A. Meeraus.

Apéndice A – Juego de datos reales

Archivo Facu.txt

```
car ca1   ano  01   cur  cal0
˝
    gru  h01_  tur 1  eqh 0
˝
       cla ct0003_lu
˝
       cla ct0003_mi
˝
       cla ct0003_ju
˝
       cla cp0020_ma
˝
       cla cp0020_vi

    gru  h02_  tur 1  eqh 0
       cla ct0003_lu
       cla ct0003_mi
       cla ct0003_ju
       cla cp0020_ma
       cla cp0020_vi

    gru  h03_  tur 1  eqh 0
       cla ct0003_lu
       cla ct0003_mi
       cla ct0003_ju
       cla cp0030_ma
       cla cp0030_ju

    gru  h04_  tur 1  eqh 0
       cla ct0003_lu
       cla ct0003_mi
       cla ct0003_ju
       cla cp0030_ma
       cla cp0030_ju

    gru  h05_  tur 1  eqh 0
       cla ct0003_lu
       cla ct0003_mi
       cla ct0003_ju
       cla cp0030_ma
       cla cp0030_ju

    gru  h06_  tur 1  eqh 0
       cla ct0003_lu
       cla ct0003_mi
       cla ct0003_ju
       cla cp0035_mi
       cla cp0035_vi

    gru  h07_  tur 1  eqh 0
       cla c1t002_ma
       cla c1t002_ju
       cla c1t002_vi
       cla c1p028_lu
       cla c1p028_mi
```

```
gru  h08_  tur 1  eqh 0
   cla c1t002_ma
   cla c1t002_ju
   cla c1t002_vi
   cla c1p028_lu
   cla c1p028_mi

gru  h09_  tur 1  eqh 0
   cla c1t002_ma
   cla c1t002_ju
   cla c1t002_vi
   cla c1p028_lu
   cla c1p028_mi

gru  h10_  tur 1  eqh 0
   cla c1t002_ma
   cla c1t002_ju
   cla c1t002_vi
   cla c1p034_lu
   cla c1p034_mi

gru  h11_  tur 1  eqh 0
   cla c1t002_ma
   cla c1t002_ju
   cla c1t002_vi
   cla c1p032_lu
   cla c1p032_ju

gru  h12_  tur 1  eqh 0
   cla c1t002_ma
   cla c1t002_ju
   cla c1t002_vi
   cla c1p032_lu
   cla c1p032_ju

gru  h13_  tur 2  eqh 0
   cla ct0004_lu
   cla ct0004_ma
   cla ct0004_ju
   cla c1p036_lu
   cla c1p036_ju

gru  h14_  tur 2  eqh 0
   cla ct0004_lu
   cla ct0004_ma
   cla ct0004_ju
   cla c1p036_lu
   cla c1p036_ju

gru  h15_  tur 2  eqh 0
   cla ct0004_lu
   cla ct0004_ma
   cla ct0004_ju
   cla c1p024_lu
   cla c1p024_mi


gru  h16_  tur 2  eqh 0
   cla ct0004_lu
   cla ct0004_ma
   cla ct0004_ju
   cla c1p024_lu
   cla c1p024_mi

gru  h17_  tur 2  eqh 0
   cla ct0004_lu
```

```
   cla ct0004_ma
   cla ct0004_ju
   cla c1p024_lu
   cla c1p024_mi

gru  h18_  tur 2  eqh 0
   cla ct0004_lu
   cla ct0004_ma
   cla ct0004_ju
   cla cp0025_ma
   cla cp0025_ju

gru  h19_  tur 2  eqh 0
   cla ct0004_lu
   cla ct0004_ma
   cla ct0004_ju
   cla cp0025_ma
   cla cp0025_ju

gru  h20_  tur 2  eqh 0
   cla ct0004_lu
   cla ct0004_ma
   cla ct0004_ju
   cla cp0025_ma
   cla cp0025_ju

gru  h21_  tur 2  eqh 0
   cla ct0005_lu
   cla ct0005_mi
   cla ct0005_ju
   cla c1p023_ma
   cla c1p023_ju

gru  h22_  tur 2  eqh 0
   cla ct0005_lu
   cla ct0005_mi
   cla ct0005_ju
   cla c1p022_ma
   cla c1p022_ju

gru  h23_  tur 2  eqh 0
   cla ct0005_lu
   cla ct0005_mi
   cla ct0005_ju
   cla cp0029_lu
   cla cp0029_mi

gru  h24_  tur 3  eqh 0
   cla ct0001_lu
   cla ct0001_mi
   cla ct0001_vi
   cla cp3133_lu
   cla cp3133_vi

gru  h25_  tur 3  eqh 0
   cla ct0001_lu
   cla ct0001_mi
   cla ct0001_vi
   cla cp2127_ma
   cla cp2127_ju

gru  h26_  tur 3  eqh 0
   cla ct0001_lu
   cla ct0001_mi
   cla ct0001_vi
   cla cp0026_ma
```

```
        cla cp0026_ju



car ca1   ano  01   cur  alg0
    gru  h01_  tur 1  eqh 0
    cla alt003_lu
    cla alt003_mi
    cla alp002_lu
    cla alp002_mi

    gru  h02_  tur 1  eqh 0
    cla alt003_lu
    cla alt003_mi
    cla alp002_lu
    cla alp002_mi

    gru  h03_  tur 1  eqh 0
    cla alt003_lu
    cla alt003_mi
    cla alp001_ma
    cla alp001_vi

    gru  h04_  tur 1  eqh 0
    cla alt003_lu
    cla alt003_mi
    cla alp001_ma
    cla alp001_vi

    gru  h05_  tur 1  eqh 0
    cla alt003_lu
    cla alt003_mi
    cla alp001_ma
    cla alp001_vi

    gru  h06_  tur 1  eqh 0
    cla alt003_lu
    cla alt003_mi
    cla alp009_ma
    cla alp009_vi

    gru  h07_  tur 1  eqh 0
    cla alt001_ma
    cla alt001_ju
    cla alp007_mi
    cla alp007_vi

    gru  h08_  tur 1  eqh 0
    cla alt001_ma
    cla alt001_ju
    cla alp007_mi
    cla alp007_vi

    gru  h09_  tur 1  eqh 0
    cla alt001_ma
    cla alt001_ju
    cla alp007_mi
    cla alp007_vi

    gru  h10_  tur 1  eqh 0
    cla alt001_ma
    cla alt001_ju
    cla alp004_mi
```

```
cla alp004_vi

 gru  h11_  tur 1  eqh 0
cla alt001_ma
cla alt001_ju
cla alp006_mi
cla alp006_vi

 gru  h12_  tur 1  eqh 0
cla alt001_ma
cla alt001_ju
cla alp006_mi
cla alp006_vi

 gru  h13_  tur 2  eqh 0
cla alt005_ma
cla alt005_ju
cla alp005_mi
cla alp005_vi

 gru  h14_  tur 2  eqh 0
cla alt005_ma
cla alt005_ju
cla alp005_mi
cla alp005_vi

 gru  h15_  tur 2  eqh 0
cla alt005_ma
cla alt005_ju
cla alp014_ma
cla alp014_vi

 gru  h16_  tur 2  eqh 0
cla alt005_ma
cla alt005_ju
cla alp014_ma
cla alp014_vi

 gru  h17_  tur 2  eqh 0
cla alt005_ma
cla alt005_ju
cla alp014_ma
cla alp014_vi

 gru  h18_  tur 2  eqh 0
cla alt005_ma
cla alt005_ju
cla alp003_mi
cla alp003_vi

 gru  h19_  tur 2  eqh 0
cla alt005_ma
cla alt005_ju
cla alp003_mi
cla alp003_vi

 gru  h20_  tur 2  eqh 0
cla alt005_ma
cla alt005_ju
cla alp003_mi
cla alp003_vi

 gru  h21_  tur 2  eqh 0
cla alt004_ma
cla alt004_vi
cla alp013_lu
```

```
    cla alp013_mi

     gru  h22_  tur 2  eqh 0
    cla alt004_ma
    cla alt004_vi
    cla alp012_lu
    cla alp012_mi

     gru  h23_  tur 2  eqh 0
    cla alt004_ma
    cla alt004_vi
    cla alp011_ma
    cla alp011_ju

     gru  h24_  tur 3  eqh 0
    cla alt002_ma
    cla alt002_ju
    cla alp010_ma
    cla alp010_ju

     gru  h25_  tur 3  eqh 0
    cla alt002_ma
    cla alt002_ju
    cla alp015_lu
    cla alp015_mi

     gru  h26_  tur 3  eqh 0
    cla alt002_ma
    cla alt002_ju
    cla alp008_ma
    cla alp008_ju




car ca1   ano  01   cur  fis0
     gru  h01_  tur 1  eqh 0
    cla fgt002_ma
    cla fgt002_ju
    cla fgp046_ma
    cla fgp046_ju

     gru  h02_  tur 1  eqh 0
    cla fgt002_ma
    cla fgt002_ju
    cla fgp046_ma
    cla fgp046_ju

     gru  h03_  tur 1  eqh 0
    cla fgt002_ma
    cla fgt002_ju
    cla fgp040_lu
    cla fgp040_mi

     gru  h04_  tur 1  eqh 0
    cla fgt002_ma
    cla fgt002_ju
    cla fgp040_lu
    cla fgp040_mi

     gru  h05_  tur 1  eqh 0
    cla fgt002_ma
    cla fgt002_ju
    cla fgp400_lu
    cla fgp400_mi
```

```
 gru  h06_  tur 1  eqh 0
cla fgt002_ma
cla fgt002_ju
cla fgp043_lu
cla fgp043_ju

 gru  h07_  tur 1  eqh 0
cla fgt001_lu
cla fgt001_mi
cla fgp047_lu
cla fgp047_ju

 gru  h08_  tur 1  eqh 0
cla fgt001_lu
cla fgt001_mi
cla fgp047_lu
cla fgp047_ju
 gru  h09_  tur 1  eqh 0
cla fgt001_lu
cla fgt001_mi
cla fgp047_lu
cla fgp047_ju

 gru  h10_  tur 1  eqh 0
cla fgt001_lu
cla fgt001_mi
cla fgp042_ma
cla fgp042_vi

 gru  h11_  tur 1  eqh 0
cla fgt001_lu
cla fgt001_mi
cla fgp041_ma
cla fgp041_vi

 gru  h12_  tur 1  eqh 0
cla fgt001_lu
cla fgt001_mi
cla fgp410_mi
cla fgp410_vi

 gru  h13_  tur 2  eqh 0
cla fgt004_lu
cla fgt004_mi
cla fgp440_mi
cla fgp440_vi

 gru  h14_  tur 2  eqh 0
cla fgt004_lu
cla fgt004_mi
cla fgp440_mi
cla fgp440_vi

 gru  h15_  tur 2  eqh 0
cla fgt004_lu
cla fgt004_mi
cla fgp044_ma
cla fgp044_vi

 gru  h16_  tur 2  eqh 0
cla fgt004_lu
cla fgt004_mi
cla fgp044_ma
cla fgp044_vi
```

```
     gru  h17_  tur 2  eqh 0
    cla fgt004_lu
    cla fgt004_mi
    cla fgp441_ma
    cla fgp441_ju

     gru  h18_  tur 2  eqh 0
    cla fgt004_lu
    cla fgt004_mi
    cla fgp480_mi
    cla fgp480_vi

     gru  h19_  tur 2  eqh 0
    cla fgt004_lu
    cla fgt004_mi
    cla fgp480_mi
    cla fgp480_vi

     gru  h20_  tur 2  eqh 0
    cla fgt004_lu
    cla fgt004_mi
    cla fgp481_vi

     gru  h21_  tur 2  eqh 0
    cla fgt003_ma
    cla fgt003_vi
    cla fgp048_lu
    cla fgp048_mi

     gru  h22_  tur 2  eqh 0
    cla fgt003_ma
    cla fgt003_vi
    cla fgp053_lu
    cla fgp053_ju

     gru  h23_  tur 2  eqh 0
    cla fgt003_ma
    cla fgt003_vi
    cla fgp051_lu
    cla fgp051_mi

     gru  h24_  tur 3  eqh 0
    cla fg1t05_ma
    cla fg1t05_ju
    cla fg1p45_lu
    cla fg1p45_ju

     gru  h25_  tur 3  eqh 0
    cla fg1t05_ma
    cla fg1t05_ju
    cla fg1p52_lu
    cla fg1p52_mi

     gru  h26_  tur 3  eqh 0
    cla fg1t05_ma
    cla fg1t05_ju
    cla fg1p49_lu
    cla fg1p49_mi




car ca1   ano  01   cur  tal0
     gru  h01_  tur 1  eqh 0
    cla tdp73a_vi
```

```
    cla tdt004_ju

    gru  h02_  tur 1  eqh 0
    cla tdp73b_vi
    cla tdt004_ju

    gru  h03_  tur 1  eqh 0
    cla tdp060_vi
    cla tdt004_ju

    gru  h04_  tur 1  eqh 0
    cla tdp066_vi
    cla tdt004_ju

    gru  h07_  tur 1  eqh 0
    cla tdp061_ma
    cla tdt004_ju

    gru  h08_  tur 1  eqh 0
    cla tdp068_ma
    cla tdt004_ju

    gru  h09_  tur 1  eqh 0
    cla tdp072_ma
    cla tdt004_ju

    gru  h13_  tur 2  eqh 0
    cla tdp65a_ma
    cla tdt003_mi

    gru  h14_  tur 2  eqh 0
    cla tdp65b_ma
    cla tdt003_mi

    gru  h15_  tur 2  eqh 0
    cla tdp069_mi
    cla tdt002_ju

    gru  h16_  tur 2  eqh 0
    cla tdp063_mi
    cla tdt002_ju

    gru  h17_  tur 2  eqh 0
    cla tdp064_vi
    cla tdt002_ju

    gru  h18_  tur 2  eqh 0
    cla tdt005_vi

    gru  h19_  tur 2  eqh 0
    cla tdt005_vi

    gru  h20_  tur 2  eqh 0
    cla tdt005_vi

    gru  h24_  tur 3  eqh 0
    cla tdt003_mi
    cla tdp062_mi



tur 1 000 012
tur 2 006 021
tur 3 016 030

cant_horas 155
```

```
cant_horas_dia 31
```

Archivo Clase.txt

```
cla  ct0003_lu  est 150   imp 10  dur 3   teo 1
˝
  ran 000 003 lu
˝

˝
  otp 0 0 0 0 0 0 0 0 0 0
˝

˝

˝
cla  ct0003_mi  est 150   imp 10  dur 3   teo 1
˝
  ran 000 003 mi
˝

˝
  otp 0 0 0 0 0 0 0 0 0 0
˝

˝

˝
cla  ct0003_ju  est 150   imp 10  dur 3   teo 1
  ran 000 003 ju

  otp 0 0 0 0 0 0 0 0 0 0


cla  cp0020_ma  est 150   imp 10  dur 3   teo 0
  ran 000 003 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  cp0020_vi  est 150   imp 10  dur 3   teo 0
  ran 006 009 vi

  otp 0 0 0 0 0 0 0 0 0 0


cla  cp0030_ma  est 150   imp 10  dur 3   teo 0
  ran 008 011 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  cp0030_ju  est 150   imp 10  dur 3   teo 0
  ran 007 010 ju

  otp 0 0 0 0 0 0 0 0 0 0


cla  cp0035_mi  est 150   imp 10  dur 3   teo 0
  ran 006 009 mi

  otp 0 0 0 0 0 0 0 0 0 0
```

```
cla  cp0035_vi  est 150   imp 10  dur 3   teo 0
  ran 003 006 vi

  otp 0 0 0 0 0 0 0 0 0 0


cla  c1t002_ma  est 150   imp 10  dur 3   teo 1
  ran 000 003 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  c1t002_ju  est 150   imp 10  dur 3   teo 1
  ran 000 003 ju

  otp 0 0 0 0 0 0 0 0 0 0


cla  c1t002_ju  est 150   imp 10  dur 3   teo 1
  ran 000 003 ju

  otp 0 0 0 0 0 0 0 0 0 0


cla  c1t002_vi  est 150   imp 10  dur 3   teo 1
  ran 000 003 vi

  otp 0 0 0 0 0 0 0 0 0 0


cla  c1p028_lu  est 150   imp 10  dur 3   teo 0
  ran 007 010 lu

  otp 0 0 0 0 0 0 0 0 0 0


cla  c1p028_mi  est 150   imp 10  dur 3   teo 0
  ran 007 010 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  c1p034_lu  est 150   imp 10  dur 3   teo 0
  ran 004 007 lu

  otp 0 0 0 0 0 0 0 0 0 0


cla  c1p034_mi  est 150   imp 10  dur 3   teo 0
  ran 007 010 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  c1p032_lu  est 150   imp 10  dur 3   teo 0
  ran 004 007 lu

  otp 0 0 0 0 0 0 0 0 0 0


cla  c1p032_ju  est 150   imp 10  dur 3   teo 0
  ran 006 009 ju

  otp 0 0 0 0 0 0 0 0 0 0
```

```
cla  ct0004_lu  est 150   imp 10  dur 3   teo 1
  ran 010 013 lu

  otp 0 0 0 0 0 0 0 0 0 0


cla  ct0004_ma  est 150   imp 10  dur 3   teo 1
  ran 010 013 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  ct0004_ju  est 150   imp 10  dur 3   teo 1
  ran 010 013 ju

  otp 0 0 0 0 0 0 0 0 0 0


cla  c1p036_lu  est 150   imp 10  dur 3   teo 0
  ran 017 020 lu

  otp 0 0 0 0 0 0 0 0 0 0


cla  c1p036_ju  est 150   imp 10  dur 3   teo 0
  ran 016 019 ju

  otp 0 0 0 0 0 0 0 0 0 0


cla  c1p024_lu  est 150   imp 10  dur 3   teo 0
  ran 017 020 lu

  otp 0 0 0 0 0 0 0 0 0 0


cla  c1p024_mi  est 150   imp 10  dur 3   teo 0
  ran 017 020 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  cp0025_ma  est 150   imp 10  dur 3   teo 0
  ran 017 020 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  cp0025_ju  est 150   imp 10  dur 3   teo 0
  ran 016 019 ju

  otp 0 0 0 0 0 0 0 0 0 0


cla  ct0005_lu  est 150   imp 10  dur 3   teo 1
  ran 013 016 lu

  otp 0 0 0 0 0 0 0 0 0 0


cla  ct0005_mi  est 150   imp 10  dur 3   teo 1
  ran 013 016 mi

  otp 0 0 0 0 0 0 0 0 0 0
```

```
cla  ct0005_ju  est 150   imp 10  dur 3   teo 1
  ran 016 019 ju

  otp 0 0 0 0 0 0 0 0 0 0


cla  c1p023_ma  est 150   imp 10  dur 3   teo 0
  ran 014 017 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  c1p023_ju  est 150   imp 10  dur 3   teo 0
  ran 013 016 ju

  otp 0 0 0 0 0 0 0 0 0 0


cla  c1p022_ma  est 150   imp 10  dur 3   teo 0
  ran 014 017 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  c1p022_ju  est 150   imp 10  dur 3   teo 0
  ran 013 016 ju

  otp 0 0 0 0 0 0 0 0 0 0


cla  cp0029_lu  est 150   imp 10  dur 3   teo 0
  ran 010 013 lu

  otp 0 0 0 0 0 0 0 0 0 0


cla  cp0029_mi  est 150   imp 10  dur 3   teo 0
  ran 010 013 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  ct0001_lu  est 150   imp 10  dur 3   teo 1
  ran 022 025 lu

  otp 0 0 0 0 0 0 0 0 0 0


cla  ct0001_mi  est 150   imp 10  dur 3   teo 1
  ran 022 025 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  ct0001_vi  est 150   imp 10  dur 3   teo 1
  ran 022 025 vi

  otp 0 0 0 0 0 0 0 0 0 0


cla  cp3133_lu  est 150   imp 10  dur 3   teo 0
  ran 025 028 lu

  otp 0 0 0 0 0 0 0 0 0 0
```

```
cla  cp3133_vi  est 150   imp 10  dur 3   teo 0
  ran 025 028 vi

  otp 0 0 0 0 0 0 0 0 0 0


cla  cp2127_ma  est 150   imp 10  dur 3   teo 0
  ran 023 026 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  cp2127_ju  est 150   imp 10  dur 3   teo 0
  ran 023 026 ju

  otp 0 0 0 0 0 0 0 0 0 0


cla  cp0026_ma  est 150   imp 10  dur 3   teo 0
  ran 027 030 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  cp0026_ju  est 150   imp 10  dur 3   teo 0
  ran 027 030 ju

  otp 0 0 0 0 0 0 0 0 0 0




cla  alt003_lu  est 150   imp 10  dur 3   teo 1
  ran 003 006 lu

  otp 0 0 0 0 0 0 0 0 0 0


cla  alt003_mi  est 150   imp 10  dur 3   teo 1
  ran 003 006 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp002_lu  est 150   imp 10  dur 3   teo 0
  ran 006 009 lu

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp002_mi  est 150   imp 10  dur 3   teo 0
  ran 007 010 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp001_ma  est 150   imp 10  dur 3   teo 0
  ran 000 003 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp001_vi  est 150   imp 10  dur 3   teo 0
  ran 006 009 vi
```

```
  otp 0 0 0 0 0 0 0 0 0 0


cla  alp009_ma  est 150   imp 10  dur 3   teo 0
  ran 000 003 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp009_vi  est 150   imp 10  dur 3   teo 0
  ran 000 003 vi

  otp 0 0 0 0 0 0 0 0 0 0


cla  alt001_ma  est 150   imp 10  dur 3   teo 1
  ran 003 006 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  alt001_ju  est 150   imp 10  dur 3   teo 1
  ran 003 006 ju

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp007_mi  est 150   imp 10  dur 3   teo 0
  ran 004 007 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp007_vi  est 150   imp 10  dur 3   teo 0
  ran 003 006 vi

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp004_mi  est 150   imp 10  dur 3   teo 0
  ran 004 007 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp004_vi  est 150   imp 10  dur 3   teo 0
  ran 003 006 vi

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp006_mi  est 150   imp 10  dur 3   teo 0
  ran 004 007 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp006_vi  est 150   imp 10  dur 3   teo 0
  ran 003 006 vi

  otp 0 0 0 0 0 0 0 0 0 0


cla  alt005_ma  est 150   imp 10  dur 3   teo 1
  ran 013 016 ma
```

```
  otp 0 0 0 0 0 0 0 0 0 0


cla  alt005_ju  est 150   imp 10  dur 3   teo 1
  ran 013 016 ju

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp005_mi  est 150   imp 10  dur 3   teo 0
  ran 010 013 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp005_vi  est 150   imp 10  dur 3   teo 0
  ran 012 015 vi

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp014_ma  est 150   imp 10  dur 3   teo 0
  ran 007 010 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp014_vi  est 150   imp 10  dur 3   teo 0
  ran 012 015 vi

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp003_mi  est 150   imp 10  dur 3   teo 0
  ran 010 013 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp003_vi  est 150   imp 10  dur 3   teo 0
  ran 012 015 vi

  otp 0 0 0 0 0 0 0 0 0 0


cla  alt004_ma  est 150   imp 10  dur 3   teo 1
  ran 017 020 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  alt004_vi  est 150   imp 10  dur 3   teo 1
  ran 015 019 vi

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp013_lu  est 150   imp 10  dur 3   teo 0
  ran 010 013 lu

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp013_mi  est 150   imp 10  dur 3   teo 0
  ran 010 013 mi
```

```
  otp 0 0 0 0 0 0 0 0 0 0


cla  alp012_lu  est 150   imp 10  dur 3   teo 0
  ran 010 013 lu

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp012_mi  est 150   imp 10  dur 3   teo 0
  ran 010 013 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp011_ma  est 150   imp 10  dur 3   teo 0
  ran 014 017 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp011_ju  est 150   imp 10  dur 3   teo 0
  ran 013 016 ju

  otp 0 0 0 0 0 0 0 0 0 0


cla  alt002_ma  est 150   imp 10  dur 3   teo 1
  ran 020 023 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  alt002_ju  est 150   imp 10  dur 3   teo 1
  ran 020 023 ju

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp010_ma  est 150   imp 10  dur 3   teo 0
  ran 023 026 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp010_ju  est 150   imp 10  dur 3   teo 0
  ran 023 026 ju

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp015_lu  est 150   imp 10  dur 3   teo 0
  ran 025 028 lu

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp015_mi  est 150   imp 10  dur 3   teo 0
  ran 025 028 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  alp008_ma  est 150   imp 10  dur 3   teo 0
  ran 024 027 ma
```

```
  otp 0 0 0 0 0 0 0 0 0 0


cla  alp008_ju  est 150   imp 10  dur 3   teo 0
  ran 024 027 ju

  otp 0 0 0 0 0 0 0 0 0 0







cla  fgt002_ma  est 150   imp 10  dur 4   teo 1
  ran 003 007 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgt002_ju  est 150   imp 10  dur 4   teo 1
  ran 003 007 ju

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp046_ma  est 150   imp 10  dur 3   teo 0
  ran 007 010 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp046_ju  est 150   imp 10  dur 3   teo 0
  ran 007 010 ju

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp040_lu  est 150   imp 10  dur 3   teo 0
  ran 006 009 lu

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp040_mi  est 150   imp 10  dur 3   teo 0
  ran 006 009 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp400_lu  est 150   imp 10  dur 3   teo 0
  ran 006 009 lu

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp400_mi  est 150   imp 10  dur 3   teo 0
  ran 006 009 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp043_lu  est 150   imp 10  dur 3   teo 0
  ran 006 009 lu
```

```
  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp043_ju  est 150   imp 10  dur 3   teo 0
  ran 007 010 ju

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgt001_lu  est 150   imp 10  dur 4   teo 1
  ran 000 004 lu

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgt001_mi  est 150   imp 10  dur 4   teo 1
  ran 000 004 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp047_lu  est 150   imp 10  dur 3   teo 0
  ran 004 007 lu

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp047_ju  est 150   imp 10  dur 3   teo 0
  ran 006 009 ju

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp042_ma  est 150   imp 10  dur 3   teo 0
  ran 006 009 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp042_vi  est 150   imp 10  dur 3   teo 0
  ran 006 009 vi

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp041_ma  est 150   imp 10  dur 3   teo 0
  ran 006 009 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp041_vi  est 150   imp 10  dur 3   teo 0
  ran 006 009 vi

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp410_mi  est 150   imp 10  dur 3   teo 0
  ran 007 010 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp410_vi  est 150   imp 10  dur 3   teo 0
  ran 006 009 vi
```

```
  otp 0 0 0 0 0 0 0 0 0 0


cla  fgt004_lu  est 150   imp 10  dur 4   teo 1
  ran 013 017 lu

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgt004_mi  est 150   imp 10  dur 4   teo 1
  ran 013 017 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp440_mi  est 150   imp 10  dur 3   teo 0
  ran 017 020 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp440_vi  est 150   imp 10  dur 3   teo 0
  ran 015 018 vi

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp044_ma  est 150   imp 10  dur 3   teo 0
  ran 016 019 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp044_vi  est 150   imp 10  dur 3   teo 0
  ran 015 018 vi

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp441_ma  est 150   imp 10  dur 3   teo 0
  ran 016 019 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp441_ju  est 150   imp 10  dur 3   teo 0
  ran 016 019 ju

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp480_mi  est 150   imp 10  dur 3   teo 0
  ran 017 020 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp480_vi  est 150   imp 10  dur 3   teo 0
  ran 015 018 vi

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp481_vi  est 150   imp 10  dur 3   teo 0
  ran 015 018 vi
```

```
  otp 0 0 0 0 0 0 0 0 0 0


cla  fgt003_ma  est 150   imp 10  dur 4   teo 1
  ran 010 014 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgt003_vi  est 150   imp 10  dur 4   teo 1
  ran 011 015 vi

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp048_lu  est 150   imp 10  dur 3   teo 0
  ran 016 019 lu

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp048_mi  est 150   imp 10  dur 3   teo 0
  ran 016 019 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp053_lu  est 150   imp 10  dur 3   teo 0
  ran 016 019 lu

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp053_ju  est 150   imp 10  dur 3   teo 0
  ran 010 013 ju

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp051_lu  est 150   imp 10  dur 3   teo 0
  ran 016 019 lu

  otp 0 0 0 0 0 0 0 0 0 0


cla  fgp051_mi  est 150   imp 10  dur 3   teo 0
  ran 016 019 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  fg1t05_ma  est 150   imp 10  dur 4   teo 1
  ran 016 019 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  fg1t05_ju  est 150   imp 10  dur 4   teo 1
  ran 016 019 ju

  otp 0 0 0 0 0 0 0 0 0 0


cla  fg1p45_lu  est 150   imp 10  dur 3   teo 0
  ran 019 022 lu
```

```
  otp 0 0 0 0 0 0 0 0 0 0


cla  fg1p45_ju  est 150   imp 10  dur 3   teo 0
  ran 026 029 ju

  otp 0 0 0 0 0 0 0 0 0 0


cla  fg1p52_lu  est 150   imp 10  dur 3   teo 0
  ran 019 022 lu

  otp 0 0 0 0 0 0 0 0 0 0


cla  fg1p52_mi  est 150   imp 10  dur 3   teo 0
  ran 019 022 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  fg1p49_lu  est 150   imp 10  dur 3   teo 0
  ran 025 028 lu

  otp 0 0 0 0 0 0 0 0 0 0


cla  fg1p49_mi  est 150   imp 10  dur 3   teo 0
  ran 025 028 mi

  otp 0 0 0 0 0 0 0 0 0 0






cla  tdt004__ju  est 100   imp 10  dur 2   teo 1
  ran 010 012 ju

  otp 0 0 0 0 0 0 0 0 0 0


cla  tdp73a__vi  est 050   imp 10  dur 6   teo 0
  ran 000 006 vi

  otp 0 0 0 0 0 0 0 0 0 0


cla  tdp73b__vi  est 050   imp 10  dur 6   teo 0
  ran 000 006 vi

  otp 0 0 0 0 0 0 0 0 0 0


cla  tdp060__vi  est 050   imp 10  dur 6   teo 0
  ran 000 006 vi

  otp 0 0 0 0 0 0 0 0 0 0


cla  tdp066__vi  est 050   imp 10  dur 6   teo 0
  ran 000 006 vi

  otp 0 0 0 0 0 0 0 0 0 0
```

```
cla  tdp061__ma  est 050   imp 10  dur 6   teo 0
  ran 006 012 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  tdp068__ma  est 050   imp 10  dur 6   teo 0
  ran 006 012 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  tdp072__ma  est 050   imp 10  dur 6   teo 0
  ran 006 012 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  tdp65a__ma  est 050   imp 10  dur 6   teo 0
  ran 016 022 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  tdt003__mi  est 100   imp 10  dur 2   teo 1
  ran 020 022 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  tdp65b__ma  est 050   imp 10  dur 6   teo 0
  ran 016 022 ma

  otp 0 0 0 0 0 0 0 0 0 0


cla  tdp069__mi  est 050   imp 10  dur 6   teo 0
  ran 006 012 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  tdt002__ju  est 100   imp 10  dur 2   teo 1
  ran 008 010 ju

  otp 0 0 0 0 0 0 0 0 0 0


cla  tdp063__mi  est 050   imp 10  dur 6   teo 0
  ran 006 012 mi

  otp 0 0 0 0 0 0 0 0 0 0


cla  tdp064__vi  est 050   imp 10  dur 6   teo 0
  ran 006 012 vi

  otp 0 0 0 0 0 0 0 0 0 0


cla  tdt005__vi  est 100   imp 10  dur 2   teo 1
  ran 010 012 vi

  otp 0 0 0 0 0 0 0 0 0 0
```

```
cla  tdp062__mi  est 050   imp 10  dur 6   teo 0
  ran 025 030 mi

  otp 0 0 0 0 0 0 0 0 0 0
```

Archivo Salón.txt

```
˝
sal  001   cap  060
˝
   otp  1 0 1 0 1 1 1 0 0 0

sal  002   cap  060
   otp  1 0 1 0 1 1 1 0 0 0

sal  007   cap  060
   otp  1 0 1 0 1 1 1 0 0 0

sal  008   cap  060
   otp  1 0 1 0 1 1 1 0 0 0

sal  101   cap  060
   otp  1 0 1 0 1 1 1 0 0 0

sal  103   cap  100
   otp  1 0 1 0 1 1 1 0 0 0

sal  105   cap  100
   otp  1 0 1 0 1 1 1 0 0 0

sal  107   cap  150
   otp  1 0 1 0 1 1 1 0 0 0

sal  108   cap  100
   otp  1 0 1 0 1 1 1 0 0 0

sal  109   cap  060
   otp  1 0 1 0 1 1 1 0 0 0

sal  110   cap  060
   otp  1 0 1 0 1 1 1 0 0 0

sal  111   cap  060
   otp  1 0 1 0 1 1 1 0 0 0

sal  112   cap  060
   otp  1 0 1 0 1 1 1 0 0 0

sal  113   cap  100
   otp  1 0 1 0 1 1 1 0 0 0

sal  115   cap  100
   otp  1 0 1 0 1 1 1 0 0 0

sal  201   cap  150
   otp  1 0 1 0 1 1 1 0 0 0

sal  202   cap  100
   otp  1 0 1 0 1 1 1 0 0 0
```

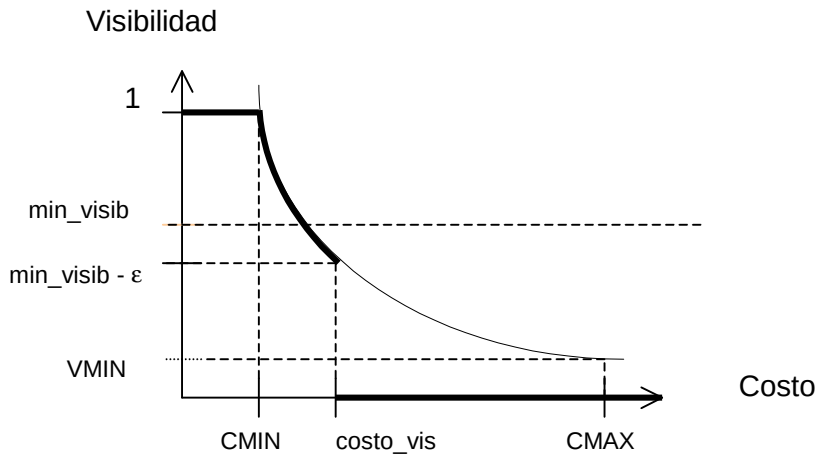
```
sal  301   cap  100
   otp  1 0 1 0 1 1 1 0 0 0

sal  401   cap  100
   otp  1 0 1 0 1 1 1 0 0 0
```

˝

˝

˝

Apéndice B – Cálculo de los coeficientes de la parábola (función de visibilidad)



Los puntos que deberán pertenecer a la parábola son :

a) (CMIN, 1)
b) (CMAX, VMIN)
c) (costo_vis, min_visib - ε)

Sean:
c_0 = CMIN
f_0 = 1
c_1 = CMAX
f_1 = VMIN
x_1 = costo_vis
f_2 = min_visib - ε

La parábola es de la forma :

$F(X)= a/X^2 + b/X + C$ para x en (c0,c1)
F0 si x=c0
F1 si x=c1

Y cumple que:
$F(c_0) = f_0$
$F(c_1) = f_1$
$F(x_1) = f_2$

Mostraremos una forma de hallar algorítmicamente a, b y c, y veremos como el método es consistente.

Resolviendo el sistema de ecuaciones:

$F(x) = a/x^2 + b/x + c$
$F(c_0)=f_0$
$F(c_1)=f_1$
F(x1)=f2

Llegamos a que

I) $A/c_0^2 + b/c_0 + c = f_0$
II) $A/c_1^2 + b/c_1 + c = f_1$
III) $A/x_1^2 + b/x_1 + c = f_2$

Hallaremos a, b y c en función de c_0, c_1, x_1, f_0, f_1, f_2

De I) y II): $a/c_0^2 + b/c_0 – f_0 = a/c_1^2 + b/c_1 - f_1$

$\Rightarrow$ $a(1/c_0^2 – 1/c_1^2) + b(1/c_0-1/c_1) = f_0 - f_1$ (*)

De II) y III): $a/c_1^2 + b/c_1 – f_1 = a/x_1^2 + b/x_1 - f_2$

$\Rightarrow$ $a(1/c_1^2 – 1/x_1^2) + b(1/c_1 -1/x_1) = f_1 - f_2$ (**)

Sean: $d_0 = 1/c_0^2 - 1/c_1^2$
$d_1 = 1/c_0 - 1/c_1$
$d_2 = f_0 - f_1$

$e_0 = 1/c_1^2 - 1/x_1^2$
$e_1 = 1/c_1 - 1/x_1$
$e_2 = f_1 - f_2$

entonces:

(*) $a \cdot d_0 + b \cdot d_1 = d_2$

(**) $a \cdot e_0 + b \cdot e_1 = e_2$

Despejando b:

$b = (d_2 + e_2) / (d_1 – d_0 \cdot e_1/e_0)$

Entonces:

$a = (d_2 - b \cdot d_1) / d_0$

$c = f_0 - a/c_0^2 - b/c_0$

recordar que $c_0 > 0$

b se puede calcular $\Leftrightarrow$ $(e_0 \neq 0)$ y $((e_0 \cdot d_1) - (d_0 \cdot e_1) \neq 0)$

Lo primero se cumple $\Leftrightarrow$ $c_0 \neq c_1$, es decir siempre.

lo segundo es

$e_0 \cdot d_1 \neq d_0 \cdot e_1$ y eso se cumple si y solo si

$$(1/c_1^2 - 1/x_1^2)(1/c_0 - 1/c_1) \neq (1/c_0^2 - 1/c_1^2)(1/c_1 - 1/x_1) \Leftrightarrow$$

$$\frac{(x_1^2 - c_1^2)(c_1 - c_0)}{c_1^2 \cdot x_1^2 \cdot c_0 \cdot c_1} \neq \frac{(c_1^2 - c_0^2)(x_1 - c_1)}{c_0^2 \cdot c_1^2 \cdot x_1 \cdot c_1} \Leftrightarrow$$

$$\frac{(x_1 - c_1)(x_1 + c_1)(c_1 - c_0)}{c_0 \cdot x_1^2 \cdot c_1^3} \neq \frac{(c_1 - c_0)(c_1 + c_0)(x_1 - c_1)}{c_0^2 \cdot c_1^3 \cdot x_1}$$

$\Leftrightarrow$ (como $c_1 \neq c_0$, $c_1 \neq x_1$ y c_0, c_1, x_1 son distintos de 0 puedo cancelar)

$$\frac{x_1 + c_1}{x_1} \neq \frac{c_1 + c_0}{c_0} \Leftrightarrow$$

$x_1 \cdot c_0 + c_1 \cdot c_0 \neq x_1 \cdot c_1 + x_1 \cdot c_0$ $\Leftrightarrow$ (como $c_1 \neq 0$ puedo cancelar)

$x_1 \neq c_0$

y esto se cumple siempre ya que estamos considerando $x_1 \in (c_0, c_1)$
($F(c_0) = f_0$ por definición).

Entonces hemos probado que *b* puede hallarse siempre.

Luego, como $d_0 \neq 0$ entonces *a* puede hallarse siempre.

Como $c_0 > 0$ entonces *c* también puede hallarse siempre

Entonces hemos encontrado un método que nos permite determinar los valores de los coeficientes de la función F en el intervalo (c_0, c_1)

Entonces la función F(x) será igual a:

f_0 si $x = c_0$
f_1 si $x = c_1$

$a/x^2 + b/x + c$ si $x \in (c_0, c_1)$

Con a, b y c calculables por el algoritmo como ya vimos.

Apéndice C – Fuentes

Fuentes del algoritmo Ant System clásico

Auxiliar.hpp

```
#include <string.h>
˝
#include <stdio.h>
˝
#include <stdlib.h>
˝

˝
#define TRUE  1
˝
#define FALSE 0
˝

˝

˝
typedef char t_string10[10];

typedef char t_string5[5];


struct rango
{
 int hor_ini , hor_fin;
};


typedef rango rangos[10];
typedef int   vectint[10];



void arrcpy1(rangos pref_hors_1,rangos pref_hors_2);

void arrcpy2(vectint arr_1,vectint arr_2);


// *******   indica si una cadena de caracteres esta no vacia ******

int  no_vacio( char identif[]  );

// ********* comparar dos cadenas de caracteres hasta largo n ******

int cmp_str( char str1[], char str2[], int n );
```

˝

Auxiliar.cpp

```
#include "auxiliar.hpp"


void arrcpy1(rangos pref_hors_1, rangos pref_hors_2)
{
 int i;
 for(i=0;i<=9;i++)
  pref_hors_1[i]=pref_hors_2[i];
}

void arrcpy2(vectint arr_1,vectint arr_2)
{
 int i;
 for(i=0;i<=9;i++)
  arr_1[i]=arr_2[i];
}

// ******   indica si una cadena de caracteres esta no vacia ******

int no_vacio(char identif[])
{
  if (identif[0]==' ')
    return FALSE;
  else
    return TRUE;

}


// ******** comparar dos cadenas de caracteres hasta largo n ******
// si son iguales devuelve 1
// si no, 0.
// no enviar ningun string nulo.

int cmp_str( char str1[], char str2[], int n )
{
    int i;

    if ((strlen(str1)<n) || (strlen(str2)<n))
    {
        if (strlen(str1) < strlen(str2))
        n = strlen(str1);
        else
        n = strlen(str2);
    }

    for(i=0;i<n;i++)
    {
        if (str1[i]!=str2[i])
            return (0);
    }

    return(1);
}
```

Cargadat.hpp

```
˝

˝
// ***************** Consumir blancos ******************
˝

˝
void consumir_blk( char buffer[] , int &i );
˝

˝
// ******************* leer identif *******************

void leer_identif( char buffer[], int &i, char ident[] );

// ******************* buscar identif *******************

int buscar_identif( char buffer[], int &i, char ident[] );

//******************* guardar carrera *******************

int guardar_carrera( char carrera[], t_carreras_fac cars_fac );

//***************** guardar anio ****************************

int  guardar_anio( char anio[], t_anios_car anios_car );

//**************** guardar curso ****************************

int  guardar_curso( char curso[], t_cursos_anio cursos_anio );

//******************** guardar grupo ******************************

void  guardar_grupo( char grupo[], int p_turno, int p_ctrl_eq_hini,
int *p_ind_grupo, t_grupos_cur grupos_cur );

//**************** guardar grupo en gru_caranio ******************

void guardar_grupo_en_gru_caranio(int ind_car, int ind_anio, t_string5
grupo,
                                  matriz3d_str &gru_caranio);

//************************ guardar clase *************************

//  clase[] se guarda en el vector de clases de facultad
//   el indice en ese vector se guarda en clases_gru.

void  guardar_clase( char carrera[], char anio[], char curso[],
                     char grupo[], char p_clase[], int ind_car,
                     int ind_anio, int ind_cur, int ind_gru,
                     int *p_ind_clase, t_clases_gru clases_gru,
                     t_clases_fac vect_clases );

//***************** guardar salon ***************************

void guardar_salon( char id_salon[],int capacidad,vectint otras_car,
                  t_salones vect_salones, int *p_ind_salon );

//******************** guardar turno ********************

void guardar_turno( int turno, int h_ini, int h_fin,
                    t_vect_turnos &v_turnos );
```

```
// *************** cargar curso ********************

int cargar_curso( FILE *fp, char carrera[], char anio[], char curso[]
);

// *************** cargar grupo ********************

int cargar_grupo( FILE *fp, char grupo[], int *p_turno, int
*p_ctrl_eq_hini );

// *************** cargar clase ********************

// Carga el identificador de la clase a partir del archivo de
// cursos.

int cargar_clase( FILE *fp, char clase[] );


// *************** cargar turno ********************

int cargar_turno( FILE *fp, int *turno, int *h_ini, int *h_fin );


// ***************** cargar_carreras *******************

void cargar_carreras ( char *arch, t_carreras_fac facultad,
                t_clases_fac &vect_clases, t_vect_turnos &v_turnos,
                matriz3d_str &gru_caranio );

//************* cargar datos clase ************************

int cargar_datos_clase( FILE *fp, char id_clase[], int *p_cant_estud,
                    int *p_importancia, int *p_duracion, int
*p_es_teorico);

//********************** cargar rangos ************************

void   cargar_rangos( FILE *fp, rangos rangos_hor );

//********************** cargar otras pref ***********************

void   cargar_otraspref( FILE *fp, vectint otras_pref );

//************ busca una id_clase en el vector de clases **********

int buscar_ind_clases_fac( char p_clase[], t_clases_fac clases_fac );

//*********** cargar clases ****************************

// Carga todos los datos de las clases a partir del archivo de clases.

void cargar_clases_facu( char *arch, t_clases_fac clases_fac  );

//********************** cargar datos salon ***********************

int cargar_datos_salon( FILE *fp, char id_salon[], int *capacidad );

//*************************  cargar salones *********************

void cargar_salones( char *arch , t_salones &vect_salones );

// ***************** Cargar nombres de archivos *******************

void cargar_noms_archivos( char *indice, char nom_arch1[13],
                                char nom_arch2[13], char nom_arch3[13],
                                char nom_arch4[13] );
```

```
//**************** Cargar valores ********************************

void cargar_valores(char buffer[100], char carrera[4], char anio[3]);

// ******************** obtener cantidad de **********************
// **** carreras, años, cursos, grupos, clases de un grupo *******

void obtener_cants_cur(char *arch_cur);

//************** Obtener cantidad de clases ******************

void obtener_cants_cla(char *arch_cla);

//************** Obtener cantidad de salones ******************

void obtener_cants_sal(char *arch_sal);

//******************** Obtener cantidades ***********************

void obtener_cants(char *arch_cur, char *arch_cla, char *arch_sal);

//**************** dimensionar estructura facultad ***************

void dimensionar_estructura_facu( t_carreras_fac &estructura_facu,
                                  int d1, int d2, int d3, int d4, int
d5);


//**************** carga de dimensiones ************************

void carga_de_dimensiones(char *arch_indice);


//**************** carga de datos ********************************

void carga_de_datos(char *arch_indice, matriz3d_str &gru_caranio);
˝
```

Cargadat.cpp

```
#include <time.h>
˜
#include <math.h>
˜
#include "auxiliar.hpp"
˜
#include "matrices.hpp"
#include "hormiga.hpp"
#include "estrfacu.hpp"
#include "cargadat.hpp"


extern int max_iter;
extern int cant_horm;

extern float exp_rastro;
extern float exp_visib;
extern float coef_evap;

extern float peso_ran_hor;
extern float peso_capac_salon;
extern float peso_otros_req;
extern float peso_desper_salon;
extern float peso_espaciamiento;
extern float peso_adyacencia;
extern float peso_compactibilidad;

extern float max_costo;
extern float mult_rastro;
extern float min_visib;
extern float max_visib;
extern float min_prob;
extern float min_rastro;


extern int cant_clases_gru;
extern int cant_grupos;
extern int cant_cursos;
extern int cant_anios;
extern int cant_carreras;
extern int cant_horas_dia;
extern int cant_dias_sem;


extern int cant_clases;
extern int cant_hors;
extern int cant_salones;

extern t_carreras_fac estructura_facu;
extern t_salones salones_fac;
extern t_clases_fac clases_fac;     //vector  con todas las clases de
facultad
extern t_vect_turnos v_turnos;


extern FILE *salida; // archivo donde se grabarán resultados.




// *************** Consumir blancos ******************
```

```
void consumir_blk( char buffer[] , int &i )
{
 while( (buffer[i]==' ')||(buffer[i]=='\t') )
    i++;
}



// ****************** leer identif ********************

void leer_identif( char buffer[], int &i, char ident[] )
{
 char aux[4];
 int j;

 for(j=0;j<=2;j++)
 {
  aux[j] = buffer[i];
  i++;
 }
 aux[3] = '\0';

 if (strcmp(aux,ident)!=0)
 {
    fprintf(salida,"Error en el formato del archivo.\n");
    exit(-1);
 }
}



// ***************** buscar identif ********************

int buscar_identif( char buffer[], int &i, char ident[] )
{
 char aux[4];
 int j;

 for(j=0;j<=2;j++)
 {
  aux[j] = buffer[i];
  i++;
 }
 aux[3] = '\0';

 if (strcmp(aux,ident)!=0)
    return (FALSE);
 else
    return (TRUE);

}


// ***************** obtener identif ********************

void obtener_identif( char buffer[], int &i, char ident[] )
{
 int j;


˝
```

```
 for(j=0;j<=2;j++)
 {
   ident[j] = buffer[i];
   i++;
 }
 ident[3] = '\0';
}

//*********guardar carrera ****************************


int guardar_carrera( char carrera[], t_carreras_fac cars_fac )
{
    // debe recorrer el vector de carreras ya que la carrera a insertar
    // puede existir (esto se debe al formato utilizado en el archivo
    // de datos de facultad).

  int i;
  i=0;
  while (( no_vacio(cars_fac[i].id_carrera)) &&
           ( !cmp_str(cars_fac[i].id_carrera,carrera,3)))
           i++;
  if (!cmp_str(cars_fac[i].id_carrera,carrera,3))
     strcpy(cars_fac[i].id_carrera,carrera);

  return i;
}


//*****************guardar anio*****************************


int  guardar_anio( char anio[], t_anios_car anios_car )
{
    // debe recorrer el vector de anios de la carrera ya que el año a
    // insertar

    // puede existir (esto se debe al formato utilizado en el archivo

    // de datos de facultad).


  int i;

  i=0;

  while (( no_vacio(anios_car[i].id_anio)) &&

           ( !cmp_str(anios_car[i].id_anio,anio,2)))
           i++;
  if (!cmp_str(anios_car[i].id_anio,anio,2))
     strcpy(anios_car[i].id_anio,anio);
  return i;
}
```

```
//****************guardar curso*****************************


int  guardar_curso( char curso[], t_cursos_anio cursos_anio )
{
    // debe recorrer el vector de cursos ya que el curso a insertar
    // puede existir (esto se debe al formato utilizado en el archivo
   // de datos de facultad).

  int i;
  i=0;
  while (( no_vacio(cursos_anio[i].id_curso)) &&
           ( !cmp_str(cursos_anio[i].id_curso,curso,4)))
           i++;
  if (!cmp_str(cursos_anio[i].id_curso,curso,4))
     strcpy(cursos_anio[i].id_curso,curso);
  fprintf(salida,"cursos_anio.id_curso de %i es %s\n",i ,
    cursos_anio[i].id_curso);

  return i;
}



//********************guardar grupo*******************************


void  guardar_grupo( char grupo[], int p_turno, int p_ctrl_eq_hini,
                           int *p_ind_grupo, t_grupos_cur grupos_cur )
{
    // no es necesario recorrer el vector de grupos ya que los mismos se
    // cargan todos de una sola vez (pertenecen a un mismo curso) debido
    // al formato del archivo de datos de facultad.

     strcpy(grupos_cur[*p_ind_grupo].id_grupo,grupo);
     grupos_cur[*p_ind_grupo].turno = p_turno;
     grupos_cur[*p_ind_grupo].ctrl_eq_hini = p_ctrl_eq_hini;
     fprintf(salida,"grupos_cur[%i].id_grupo: %s  ",*p_ind_grupo ,
        grupos_cur[*p_ind_grupo].id_grupo);
     fprintf(salida,"  guarde  turno: %i",p_turno);
     fprintf(salida,"  guarde  ctrl_eq_hini: %i\n",p_ctrl_eq_hini);

     (*p_ind_grupo)++;

}



//***************** guardar grupo en gru_caranio *************************

void guardar_grupo_en_gru_caranio(int ind_car, int ind_anio, t_string5
                                     grupo, matriz3d_str &gru_caranio)
{
    // se recorre el vector de grupos para esa carrera y año, y si el grupo
    // no existe se inserta.

    int gru;
    int fin_grupos, esta;
    t_string5 id_grupo;

    gru = 0;
    fin_grupos = FALSE;
    esta = FALSE;
```

```
    while((!esta) && (!fin_grupos))
    {
        gru_caranio.consultar(ind_car, ind_anio, gru, id_grupo);
        if(strlen(id_grupo)==0)
        fin_grupos = TRUE;
        else
        if(cmp_str(id_grupo, grupo, 4))
               esta = TRUE;

       if ( (!fin_grupos) && (!esta) )
            gru++;
    }


    if(!esta)
    {
        gru_caranio.insertar(ind_car, ind_anio, gru, grupo);
    }

}



//************************guardar clase*********************************



void  guardar_clase( char carrera[], char anio[], char curso[],
                        char grupo[], char p_clase[],int ind_car,
                        int ind_anio, int ind_cur, int ind_gru,
                        int *p_ind_clase,t_clases_gru clases_gru,
                        t_clases_fac vect_clases )
{
  int i;
  t_vect_ubicacion v_ubic;

  // clase[] se guarda en el vector de clases de facultad
  // el indice en ese vector se guarda en clases_gru.

  // si no existe insertar la clase en clases_fac
  // si existe agregar la nueva ubicacion de la clase.
  // devolver el indice de clases_gru incrementado (proximo lugar libre).

  i=0;
  while ((!cmp_str( vect_clases[i].id_clase ,p_clase, 9)) &&
                  (!cmp_str( vect_clases[i].id_clase," ", 1)))
        {
             i++;
        }

  if (cmp_str( vect_clases[i].id_clase, " ", 1 ))
  {
        // insertar nueva clase en clases_fac:
        vect_clases[i]=clase( p_clase );
        fprintf(salida,"\nGuarde vect_clases[%i]: %s\n",i,
               vect_clases[i].id_clase);
˜
        // guardar ubicacion de la clase insertada:
˜
        vect_clases[i].guardar_ubicacion(carrera, anio, curso, grupo,
˜
                                         ind_car, ind_anio, ind_cur, ind_gru);

        vect_clases[i].consultar_ubicacion(v_ubic);
        fprintf(salida,"guarde ubicacion en v_ubic[0]: car: %s, anio: %s,
```

```
                    cur: %s, gru: %s \n", v_ubic.vect[0].carrera,
                    v_ubic.vect[0].anio, v_ubic.vect[0].curso,
                    v_ubic.vect[0].grupo);
  }
  else
  {
    // agregar nueva ubicacion a una clase ya existente en clases_fac:


    vect_clases[i].guardar_ubicacion(carrera, anio, curso, grupo,
                                        ind_car, ind_anio, ind_cur, ind_gru);
    fprintf(salida,"agregue otra ubicacion para vect_clases de %i : %s\n",i,
              vect_clases[i].id_clase);
    vect_clases[i].consultar_ubicacion(v_ubic);
    fprintf(salida,"   ubicacion en v_ubic[%i]: car: %s, anio: %s, cur: %s,
           gru: %s \n",v_ubic.tope-1, v_ubic.vect[v_ubic.tope-1].carrera,
           v_ubic.vect[v_ubic.tope-1].anio, v_ubic.vect[v_ubic.tope-1].curso,
           v_ubic.vect[v_ubic.tope-1].grupo);

  }


   // insertar indice en clases_gru[*p_ind_clase]

    // no es necesario recorrer el vector de clases de un grupo ya que
    // las mismas se cargan todas de una sola vez debido
    // al formato del archivo de datos de facultad.

  clases_gru[*p_ind_clase]=i;
  (*p_ind_clase)++;

}


//*****************guardar salon***************************

void guardar_salon( char id_salon[],int capacidad,vectint otras_car,
                   t_salones salones_fac, int *p_ind_salon )
{
     salones_fac[*p_ind_salon]=salon(id_salon,capacidad,otras_car);
     fprintf(salida,"salones_fac[%i].id_salon: %s\n", *p_ind_salon,
           salones_fac[*p_ind_salon].id_salon);

     fprintf(salida,"               .capacidad: %i\n",
              salones_fac[*p_ind_salon].capacidad);


     (*p_ind_salon)++;
}


//******************** guardar turno ******************************
```

```
void  guardar_turno( int p_turno, int p_h_ini, int p_h_fin,
                         t_vect_turnos &v_turnos )
{
    v_turnos.vect[v_turnos.tope].turno=p_turno;
    v_turnos.vect[v_turnos.tope].h_ini=p_h_ini;
    v_turnos.vect[v_turnos.tope].h_fin=p_h_fin;
    v_turnos.tope++;
}



// *************** cargar curso *********************

int cargar_curso( FILE *fp, char carrera[], char anio[], char curso[] )
{
 char buffer[100];
 char identif[4];
 int i, j;
 char *fin;
 long offset;  // offset dentro del archivo.
 int es_car, es_tur;

 do
 {
    i=0;
    offset = ftell(fp);  // guardo posicion del puntero del archivo.
    fin = fgets(buffer,100,fp);
    if (!(fin==NULL))
        {
            consumir_blk(buffer,i);
            obtener_identif(buffer,i,identif);
            es_car = cmp_str(identif,"car",3);
            es_tur = cmp_str(identif,"tur",3);
        }
 }
 while((!(fin==NULL))&&(!es_car)&&(!es_tur));




 if(fin==NULL)
     return(TRUE);

 if(es_tur)
 {
    fseek(fp, offset, SEEK_SET); // vuelvo a posicionar el puntero del
                                  // archivo al comienzo de la linea
                                 // que acabo de leer.
    return (TRUE);
 }
 else
 {
    i=0;
    consumir_blk(buffer,i);

    leer_identif(buffer,i,"car");

    consumir_blk(buffer,i);

    for(j=0;j<=2;j++)
    {
        carrera[j]=buffer[i];
        i++;
    }
    carrera[3]='\0';
```

```
   consumir_blk(buffer,i);

   leer_identif(buffer,i,"ano");

   consumir_blk(buffer,i);

   for(j=0;j<=1;j++)
   {
       anio[j]=buffer[i];
       i++;
   }
   anio[2]='\0';

   consumir_blk(buffer,i);

   leer_identif(buffer,i,"cur");

   consumir_blk(buffer,i);

   for(j=0;j<=3;j++)
   {
    curso[j]=buffer[i];
    i++;
   }
   curso[4]='\0';

   return(FALSE);
 }
}



// *************** cargar grupo *********************

int cargar_grupo( FILE *fp, char grupo[], int *p_turno,
                                        int *p_ctrl_eq_hini )
{
 char buffer[100];
 char identif[4];
 char turno_c[2];
 char eqh_c[2];
 int i, j;
 char *fin;
 long offset;  // offset dentro del archivo.
 int es_car, es_gru, es_tur;


 do
 {
   i=0;
  offset = ftell(fp);  // guardo posicion del puntero del archivo.
  fin = fgets(buffer,100,fp);
   if (!(fin==NULL))
       {
           consumir_blk(buffer,i);
           obtener_identif(buffer,i,identif);
           es_car = cmp_str(identif,"car",3);
           es_gru = cmp_str(identif,"gru",3);
           es_tur = cmp_str(identif,"tur",3);
       }
 }
 while((!(fin==NULL))&&(!es_car)&&(!es_gru)&&(!es_tur));
```

```
 if(fin==NULL)
     return(TRUE);

 if((es_car)||(es_tur))
 {
    fseek(fp, offset, SEEK_SET); // vuelvo a posicionar el puntero del
                                   // archivo al comienzo de la linea
                                 // que acabo de leer.
    return (TRUE);
 }
 else
 {

    // cargo nombre del grupo:

    i=0;
    consumir_blk(buffer,i);

    leer_identif(buffer,i,"gru");

    consumir_blk(buffer,i);

    for(j=0;j<=3;j++)
    {
        grupo[j]=buffer[i];
        i++;
    }
    grupo[4]='\0';

   // cargo turno del grupo:

    consumir_blk(buffer,i);
    leer_identif(buffer,i,"tur");
    consumir_blk(buffer,i);

    turno_c[0]=buffer[i];
    i++;
    turno_c[1]='\0';

    *p_turno=atoi(turno_c);


     // cargo ctrl_eq_hini del grupo:

    consumir_blk(buffer,i);
    leer_identif(buffer,i,"eqh");
    consumir_blk(buffer,i);

    eqh_c[0]=buffer[i];
    i++;
    eqh_c[1]='\0';

   *p_ctrl_eq_hini=atoi(eqh_c);


    return(FALSE);
 }
}


// *************** cargar clase *********************

// Carga el identificador de la clase a partir del archivo de
// cursos.

int cargar_clase( FILE *fp, char clase[] )
```

```
{
 char buffer[100];
 int i, j;
 char *fin;
 char identif[4];
 long offset;  // offset dentro del archivo.
 int es_car, es_gru, es_cla, es_tur;



 do
 {
   i=0;
   offset = ftell(fp);  // guardo posicion del puntero del archivo.
   fin = fgets(buffer,100,fp);
    if (!(fin==NULL))
        {
            consumir_blk(buffer,i);
            obtener_identif(buffer,i,identif);
            es_car = cmp_str(identif,"car",3);
            es_gru = cmp_str(identif,"gru",3);
            es_cla = cmp_str(identif,"cla",3);
            es_tur = cmp_str(identif,"tur",3);

        }
 }
 while((!(fin==NULL))&&(!es_car)&&(!es_gru)&&(!es_cla)&&(!es_tur));



 if(fin==NULL)
     return(TRUE);

 if((es_car)||(es_gru)||(es_tur))
 {
   fseek(fp, offset, SEEK_SET); // vuelvo a posicionar el puntero del
                                  // archivo al comienzo de la linea
                                // que acabo de leer.
   return (TRUE);
 }
 else
 {
   i=0;
   consumir_blk(buffer,i);

   leer_identif(buffer,i,"cla");

   consumir_blk(buffer,i);

   for(j=0;j<=8;j++)
   {
       clase[j]=buffer[i];
       i++;
   }
   clase[9]='\0';

   return(FALSE);
 }
}



//**************** cargar horas *********************
// carga la cantidad de 1/2 horas total y en un dia.
```

```
void cargar_horas(FILE *fp)
{
     char buffer[100];
     char *fin;
     long offset;
     int i, es_can;
     char identif[4];


     do
     {
        i=0;
        offset = ftell(fp);  // guardo posicion del puntero del archivo.
        fin = fgets(buffer,100,fp);
        if (!(fin==NULL))
        {
            consumir_blk(buffer,i);
            obtener_identif(buffer,i,identif);
            es_can = cmp_str(identif,"can",3);
        }
    }
    while((!(fin==NULL))&&(!es_can));


    if(fin==NULL)
    {
        fprintf(stderr,"ERROR: Falta cantidad de 1/2 horas total en el
                   archivo\n");
        fprintf(stderr,"       de datos de facultad \n");
        exit(-1);
    }


    if(es_can)
    {
        fseek(fp, offset, SEEK_SET); // vuelvo a posicionar el puntero del
                                       // archivo al comienzo de la linea
                                     // que acabo de leer.
        fin = fgets(buffer,100,fp);

        sscanf(buffer," %*s %i", &cant_hors);
        printf("\ncargue cantidad de 1/2 horas total: %i\n",cant_hors);

        fin = fgets(buffer,100,fp);
        if(fin==NULL)
        {
            fprintf(stderr,"ERROR: Falta cantidad de 1/2 horas por dia en
                       el\n");
            fprintf(stderr,"       archivo de datos de facultad \n");
            exit(-1);
        }
        sscanf(buffer," %*s %i", &cant_horas_dia);
        printf("cargue cantidad de 1/2 horas en un dia:
```

```
              %i\n",cant_horas_dia);
    }

}

// *************** cargar turno *********************

int cargar_turno( FILE *fp, int *turno, int *h_ini, int *h_fin )
{
 char buffer[100];
 char identif[4];
 char turno_c[2];
 char h_ini_c[4], h_fin_c[4];
 int i, j;
 char *fin;
 int es_tur, es_can;


 do
 {
   i=0;
   fin = fgets(buffer,100,fp);
   if (!(fin==NULL))
       {
           consumir_blk(buffer,i);
           obtener_identif(buffer,i,identif);
           es_tur = cmp_str(identif,"tur",3);
           es_can = cmp_str(identif,"can",3);
       }
 }
 while((!(fin==NULL))&&(!es_tur)&&(!es_can));



 if((fin==NULL)||(es_can))
   return(TRUE);
 else
 {
   // cargo numero del turno:

   consumir_blk(buffer,i);

   turno_c[0]=buffer[i];
   turno_c[1]='\0';
  i++;

   consumir_blk(buffer,i);

   // cargo horario inicial:

   for(j=0;j<3;j++)
```

```
    {
        h_ini_c[j]=buffer[i];
        i++;
    }
    h_ini_c[3]='\0';

   consumir_blk(buffer,i);

    // cargo horario final:

    for(j=0;j<3;j++)
    {
        h_fin_c[j]=buffer[i];
        i++;
    }
    h_fin_c[3]='\0';

    *turno=atoi(turno_c);
    *h_ini=atoi(h_ini_c);
    *h_fin=atoi(h_fin_c);

    return(FALSE);
 }
}



// *************** cargar_carreras ******************

void cargar_carreras ( char *arch, t_carreras_fac facultad,
                t_clases_fac &vect_clases, t_vect_turnos &v_turnos,
                matriz3d_str &gru_caranio )
{
 FILE *fp;

 int i, j;
 int fin_cursos, fin_grupos, fin_clases;
 char carrera[4];
 char anio[3];
 char curso[5];
 char grupo[5];
 char clase[10];
 int ind_car , ind_anio , ind_cur, ind_gru, ind_cla;
 int turno;
 int ctrl_eq_hini;

 fprintf(salida,"\n************************************\n");
 fprintf(salida,"***** carga de estructura facu ******\n");
 fprintf(salida,"************************************\n");


 fp=fopen(arch,"r");

 if (!fp)
 {
    fprintf(stderr,"ERROR:Archivo %s no encontrado\n",arch);
    exit(-1);
 }


 fin_cursos = cargar_curso(fp, carrera, anio, curso);

 if (fin_cursos)
 {
    fprintf(stderr,"ERROR:Faltan datos en el archivo %s \n",arch);
```

```
   exit(-1);
 }
 else
 {
    while(!fin_cursos)
    {
     ind_car= guardar_carrera( carrera , facultad);
     fprintf(salida,"\n***** Guarde carrera %s\n",carrera);
     ind_anio=guardar_anio( anio , facultad[ind_car].anios_car );
     fprintf(salida,"cargue anio %s\n",anio);
     ind_cur=guardar_curso( curso,
                 facultad[ind_car].anios_car[ind_anio].cursos_anio);
˝
     fprintf(salida,"cargue curso %s\n",curso);

     ind_gru=0;


     fin_grupos = cargar_grupo(fp, grupo, &turno, &ctrl_eq_hini);
     if (fin_grupos)
     {
       fprintf(stderr,"ERROR:Faltan datos en el archivo %s \n",arch);
      exit(-1);
     }
     else
     {
       while ( !fin_grupos )
       {
        guardar_grupo( grupo, turno, ctrl_eq_hini, &ind_gru,
                        facultad[ind_car].anios_car[ind_anio].
                        cursos_anio[ind_cur].grupos_cur);
˝
        guardar_grupo_en_gru_caranio(ind_car, ind_anio, grupo, gru_caranio);
        ind_cla=0;

        fin_clases = cargar_clase(fp, clase);
        if (fin_clases)
        {
       fprintf(stderr,"ERROR:Faltan datos en el archivo %s \n",arch);
         exit(-1);
       }
        else
       {
         while (!fin_clases)
        {
           guardar_clase( carrera, anio, curso, grupo, clase ,ind_car,
                        ind_anio, ind_cur,(ind_gru-1), &ind_cla,
                        facultad[ind_car].anios_car[ind_anio].
               cursos_anio[ind_cur].grupos_cur[(ind_gru-1)].clases_gru,
˝
                                vect_clases );
˝
           fin_clases=cargar_clase(fp, clase);
         }//while_cla
        }//else_fin
        fin_grupos = cargar_grupo(fp, grupo, &turno, &ctrl_eq_hini);
       }//while_gru
     }//else_fin
      fin_cursos = cargar_curso(fp, carrera, anio, curso);
    }//while_cur

 }//else_fin


 //******** cargo turnos: *********
```

```
 int fin_turnos;
 int hor_ini, hor_fin;

 fin_turnos = cargar_turno(fp, &turno, &hor_ini, &hor_fin);

 if (fin_turnos)
 {
    fprintf(stderr,"ERROR:Faltan turnos en el archivo %s \n",arch);
    exit(-1);
 }
 else
 {
     while(!fin_turnos)
     {
       guardar_turno( turno, hor_ini, hor_fin, v_turnos);
       fin_turnos=cargar_turno(fp, &turno, &hor_ini, &hor_fin );
     }
 }

 fprintf(salida,"\n****** Turnos: *****\n");
 for(i=0;i<v_turnos.tope;i++)
    fprintf(salida,"Turno[%i]: id: %i  h_ini %i h_fin %i \n",i,
               v_turnos.vect[i].turno,
               v_turnos.vect[i].h_ini, v_turnos.vect[i].h_fin);


 fclose(fp);


}

//************* cargar datos clase************************


int cargar_datos_clase( FILE *fp, char id_clase[], int *p_cant_estud,
                    int *p_importancia, int *p_duracion, int *p_es_teorico)
{
 char buffer[100];
 int i, j;
 char cant_estud_c[4];
 char imp_c[3];
 char duracion_c[2];
 char es_teo_c[2];
 char *fin;
 int es_cla;
 char identif[4];


 do
 {
    i=0;
   fin = fgets(buffer,100,fp);
    if (!(fin==NULL))
        {
            consumir_blk(buffer,i);
            obtener_identif(buffer,i,identif);
            es_cla = cmp_str(identif,"cla",3);
        }
 }
 while((!(fin==NULL))&&(!es_cla));


 if(fin==NULL)
     return(TRUE);
 else
 {
```

```
i=0;
consumir_blk(buffer,i);

leer_identif(buffer,i,"cla");

consumir_blk(buffer,i);

for(j=0;j<=8;j++)
{
    id_clase[j]=buffer[i];
    i++;
}
id_clase[9]='\0';

consumir_blk(buffer,i);

leer_identif(buffer,i,"est");

consumir_blk(buffer,i);

for(j=0;j<=2;j++)
{
    cant_estud_c[j]=buffer[i];
    i++;
}
cant_estud_c[3]='\0';

consumir_blk(buffer,i);

leer_identif(buffer,i,"imp");

consumir_blk(buffer,i);

for(j=0;j<=1;j++)
{
 imp_c[j]=buffer[i];
 i++;
}
imp_c[2]='\0';

consumir_blk(buffer,i);

leer_identif(buffer,i,"dur");

consumir_blk(buffer,i);

duracion_c[0]=buffer[i];
i++;
 duracion_c[1]='\0';

consumir_blk(buffer,i);

leer_identif(buffer,i,"teo");

consumir_blk(buffer,i);

es_teo_c[0]=buffer[i];
i++;
 es_teo_c[1]='\0';


 *p_cant_estud=atoi(cant_estud_c);
 *p_importancia=atoi(imp_c);
 *p_duracion=atoi(duracion_c);
 *p_es_teorico=atoi(es_teo_c);
```

```
   return(FALSE);
 }

}



//**********************cargar rangos*************************


void   cargar_rangos( FILE *fp, rangos rangos_hor )
{
 char buffer[100];
 int i, j ,ind_rangos, fin_rangos;
 char hor_ini_c[4];
 char hor_fin_c[4];
 char *fin;
 int dia;
 int es_ran, es_otp;
 char identif[4];
 long offset;  // offset dentro del archivo.


 // inicializo rangos:
 for(i=0;i<10;i++)
 {
   rangos_hor[i].hor_ini=0;
    rangos_hor[i].hor_fin=0;
 }

 ind_rangos=0;
 fin_rangos=FALSE;

 while(!fin_rangos)
 {

    do
    {
     i=0;
     offset = ftell(fp);  // guardo posicion del puntero del archivo.
     fin = fgets(buffer,100,fp);
        if (!(fin==NULL))
            {
                consumir_blk(buffer,i);
                obtener_identif(buffer,i,identif);
                es_ran = cmp_str(identif,"ran",3);
                es_otp = cmp_str(identif,"otp",3);

            }
    }
    while((!(fin==NULL))&&(!es_ran)&&(!es_otp));



    if(fin==NULL)
        fin_rangos=TRUE;
    else
    {
       if(es_otp)
        {
            fseek(fp, offset, SEEK_SET); // vuelvo a posicionar el puntero del
                                         // archivo al comienzo de la linea
                                       // que acabo de leer.
            fin_rangos=TRUE;
```

```
}
else
{
    i=0;
    consumir_blk(buffer,i);

    leer_identif(buffer,i,"ran");

   consumir_blk(buffer,i);

    for(j=0;j<=2;j++)
    {
        hor_ini_c[j]=buffer[i];
        i++;
    }
    hor_ini_c[3]='\0';

    rangos_hor[ind_rangos].hor_ini=atoi(hor_ini_c);


   consumir_blk(buffer,i);


    for(j=0;j<=2;j++)
    {
        hor_fin_c[j]=buffer[i];
       i++;
    }
   hor_fin_c[3]='\0';

   rangos_hor[ind_rangos].hor_fin=atoi(hor_fin_c);

    consumir_blk(buffer,i);

    obtener_identif( buffer,i, identif );

    dia=0;

    if (identif[0]=='l')
        dia=0;
    else
        if (identif[0]=='j')
            dia=3;
    else
    if (identif[0]=='v')
            dia=4;
    else
        if (identif[0]=='m')
    {
            if (identif[1]=='a')
                dia=1;
            else
                if (identif[1]=='i')
                    dia=2;
      }


    rangos_hor[ind_rangos].hor_ini=(rangos_hor[ind_rangos].hor_ini)
                                        + dia*cant_horas_dia;

    rangos_hor[ind_rangos].hor_fin=(rangos_hor[ind_rangos].hor_fin)
                                        + dia*cant_horas_dia;


    identif[2] = '\0';
```

```
           fprintf(salida,
               "Rangos horarios:  Dia: %s   Hor_ini: %i   Hor_fin: %i \n",
¨
                       identif, rangos_hor[ind_rangos].hor_ini,
¨
                       rangos_hor[ind_rangos].hor_fin );
¨

¨
           ind_rangos++;
¨
       }  //else
¨
   } //else
¨
 } // while
}


//********************** cargar otras pref***********************


void   cargar_otraspref( FILE *fp, vectint otras_pref )
{
 char buffer[100];
 int i, j, ind_pref;
 char pref_c[2];
 char *fin;
 long offset;  // offset dentro del archivo.
 int es_cla, es_otp;
 char identif[4];



 // inicializo vector de otras preferencias:
 for(i=0;i<10;i++)
    otras_pref[i]=0;

 do
 {
     i=0;
     offset = ftell(fp);  // guardo posicion del puntero del archivo.
     fin = fgets(buffer,100,fp);
        if (!(fin==NULL))
            {
                consumir_blk(buffer,i);
                obtener_identif(buffer,i,identif);
                es_cla = cmp_str(identif,"cla",3);
                es_otp = cmp_str(identif,"otp",3);

            }
 }
 while((!(fin==NULL))&&(!es_cla)&&(!es_otp));


 if(fin==NULL)
    return;

 if (es_cla)
 {
    fseek(fp, offset, SEEK_SET); // vuelvo a posicionar el puntero del
                                        // archivo al comienzo de la linea
                                    // que acabo de leer.
    return;
 }
 else
```

```
 {
    i=0;
    consumir_blk(buffer,i);

    leer_identif(buffer,i,"otp");

    fprintf(salida,"Otras preferencias: ");

    for(ind_pref=0;ind_pref<=9;ind_pref++)
    {
    consumir_blk(buffer,i);

        pref_c[0]=buffer[i];
        i++;

        pref_c[1]='\0';

        otras_pref[ind_pref]=atoi(pref_c);

        fprintf(salida," %i ", otras_pref[ind_pref]);
    }
    fprintf(salida,"\n");

 }
}



int buscar_ind_clases_fac( char p_clase[], t_clases_fac clases_fac )
{
  int indice;

  indice=0;
  while (!cmp_str(clases_fac[indice].id_clase, p_clase, 9))
        indice++;
  return (indice);
}







//***********cargar clases facu*****************************

// Carga todos los datos de las clases a partir del archivo de clases.

void cargar_clases_facu( char *arch, t_clases_fac clases_fac  )
{
 FILE *fp;
 int i, j;
 int fin_clases;
 char id_clase[10];
 int cant_estud, importancia, duracion, es_teorico;
 rangos rangos_hor;
 vectint otras_pref;
 int ind_clases_fac;

 fprintf(salida,"************************************\n");
 fprintf(salida,"***** carga de datos de clases ******\n");
 fprintf(salida,"************************************\n");


 fp=fopen(arch,"r");
```

```
 if (!fp)
 {
    fprintf(stderr,"ERROR:Archivo %s no encontrado\n",arch);
   exit(-1);
 }

 fin_clases = cargar_datos_clase( fp, id_clase, &cant_estud,
                                  &importancia, &duracion, &es_teorico );
 if (fin_clases)
 {
    fprintf(stderr,"ERROR:Faltan datos en el archivo %s \n",arch);
    exit(-1);
 }
 else
 {
   while(!fin_clases)
    {
     fprintf(salida,"\nCLASE: %s\n",id_clase);
     cargar_rangos(fp, rangos_hor);
     cargar_otraspref(fp, otras_pref);
     ind_clases_fac=buscar_ind_clases_fac( id_clase, clases_fac );
     clases_fac[ind_clases_fac].guardar_datos_clase(cant_estud,
                importancia,duracion,es_teorico,rangos_hor,otras_pref );
     fprintf(salida,"clases_fac[%i].estud : %i\n", ind_clases_fac,
                clases_fac[ind_clases_fac].cant_estud);
˜
     fprintf(salida,"              .durac : %i\n",
               clases_fac[ind_clases_fac].duracion);
˜
     fprintf(salida,"              .esteo : %i\n",
               clases_fac[ind_clases_fac].es_teorico);
˜
     fin_clases = cargar_datos_clase( fp, id_clase, &cant_estud,
˜
                                  &importancia, &duracion, &es_teorico );
˜
    }
˜
 }
˜

˜
 fclose(fp);
˜

˜
}




//**********************cargar datos salon***********************

int cargar_datos_salon( FILE *fp, char id_salon[], int *capacidad )
{
 char buffer[100];
 int i, j;
 char capacidad_c[4];
 char *fin;
 int es_sal, es_otp;
 char identif[4];
 long offset;
```

```
 do
 {
     i=0;
     offset = ftell(fp);  // guardo posicion del puntero del archivo.
     fin = fgets(buffer,100,fp);
     if (!(fin==NULL))
           {
               consumir_blk(buffer,i);
               obtener_identif(buffer,i,identif);
               es_sal = cmp_str(identif,"sal",3);
               es_otp = cmp_str(identif,"otp",3);

           }
 }
 while((!(fin==NULL))&&(!es_sal)&&(!es_otp));


 if(fin==NULL)
    return(TRUE);

 if (es_otp)
 {
    fseek(fp, offset, SEEK_SET); // vuelvo a posicionar el puntero del
                                        // archivo al comienzo de la linea
                                    // que acabo de leer.
    return(TRUE);
 }
 else
 {
    i=0;
    consumir_blk(buffer,i);
    leer_identif(buffer,i,"sal");
    consumir_blk(buffer,i);

    for(j=0;j<=2;j++)
    {
      id_salon[j]=buffer[i];
      i++;
    }
    id_salon[3]='\0';

    consumir_blk(buffer,i);

    leer_identif(buffer,i,"cap");

    consumir_blk(buffer,i);

    for(j=0;j<=2;j++)
    {
     capacidad_c[j]=buffer[i];
     i++;
    }
    capacidad_c[3]='\0';


    *capacidad=atoi(capacidad_c);

    return(FALSE);
 }

}


//*************************  cargar salones **********************
```

```
void cargar_salones( char *arch , t_salones &salones_fac )
{
 FILE *fp;

 int i, j;
 int fin_salones;
 char id_salon[4] ;
 int capacidad;
 vectint otras_car;
 int ind_salon;

 fprintf(salida,"\n************************************\n");
 fprintf(salida,"********* carga de salones **********\n");
 fprintf(salida,"************************************\n");


 fp=fopen(arch,"r");

 if (!fp)
 {
    fprintf(stderr,"ERROR:Archivo %s no encontrado\n",arch);
   exit(-1);
 }

 fin_salones = cargar_datos_salon(fp, id_salon, &capacidad );
 if (fin_salones)
 {
    fprintf(stderr,"ERROR:Faltan datos en el archivo %s \n",arch);
    exit(-1);
 }
 else
 {
   ind_salon=0;
    while (!fin_salones)
    {
     cargar_otraspref(fp, otras_car);
     guardar_salon(id_salon,capacidad,otras_car,salones_fac,&ind_salon);
     fin_salones = cargar_datos_salon(fp, id_salon, &capacidad );
   }
 }

 fclose(fp);

}



//*************************  cargar variables globales
**********************


 int cargar_vars_globales(FILE *fp)
 {
     char buffer[100];
     char *fin;
     int cargo;

     fin = fgets(buffer,100,fp);
     if(fin==NULL)
        return(FALSE);
     cargo = sscanf(buffer," %*s %i", &max_iter);
     printf("cargue max_iter: %i\n",max_iter);

     fin = fgets(buffer,100,fp);
     if(fin==NULL)
        return(FALSE);
```

```
cargo = sscanf(buffer," %*s %i", &cant_horm);
printf("cargue cant_horm: %i\n",cant_horm);


fin = fgets(buffer,100,fp);
if(fin==NULL)
   return(FALSE);
cargo = sscanf(buffer," %*s %f", &exp_rastro);
printf("cargue exp_rastro: %.2f\n",exp_rastro);

fin = fgets(buffer,100,fp);
if(fin==NULL)
   return(FALSE);
cargo = sscanf(buffer," %*s %f", &exp_visib);
printf("cargue exp_visib: %.2f\n",exp_visib);

fin = fgets(buffer,100,fp);
if(fin==NULL)
   return(FALSE);
cargo = sscanf(buffer," %*s %f", &coef_evap);
printf("cargue coef_evap: %.2f\n",coef_evap);


fin = fgets(buffer,100,fp);
if(fin==NULL)
   return(FALSE);
cargo = sscanf(buffer," %*s %f", &peso_ran_hor);
printf("cargue peso_ran_hor: %.2f\n",peso_ran_hor);

fin = fgets(buffer,100,fp);
if(fin==NULL)
   return(FALSE);
cargo = sscanf(buffer," %*s %f", &peso_capac_salon);
printf("cargue peso_capac_salon: %.2f\n",peso_capac_salon);

fin = fgets(buffer,100,fp);
if(fin==NULL)
   return(FALSE);
cargo = sscanf(buffer," %*s %f", &peso_otros_req);
printf("cargue peso_otros_req: %.2f\n",peso_otros_req);

fin = fgets(buffer,100,fp);
if(fin==NULL)
   return(FALSE);
cargo = sscanf(buffer," %*s %f", &peso_desper_salon);
printf("cargue peso_desper_salon: %.2f\n",peso_desper_salon);

fin = fgets(buffer,100,fp);
if(fin==NULL)
   return(FALSE);
cargo = sscanf(buffer," %*s %f", &peso_espaciamiento);
printf("cargue peso_espaciamiento: %.2f\n",peso_espaciamiento);

fin = fgets(buffer,100,fp);
if(fin==NULL)
   return(FALSE);
cargo = sscanf(buffer," %*s %f", &peso_adyacencia);
printf("cargue peso_adyacencia: %.2f\n",peso_adyacencia);

fin = fgets(buffer,100,fp);
if(fin==NULL)
   return(FALSE);
cargo = sscanf(buffer," %*s %f", &peso_compactibilidad);
printf("cargue peso_compactibilidad: %.2f\n",peso_compactibilidad);

fin = fgets(buffer,100,fp);
```

```
    if(fin==NULL)
       return(FALSE);
    cargo = sscanf(buffer," %*s %f", &max_costo);
    printf("cargue max_costo: %.2f\n",max_costo);

    fin = fgets(buffer,100,fp);
    if(fin==NULL)
       return(FALSE);
    cargo = sscanf(buffer," %*s %f", &mult_rastro);
    printf("cargue mult_rastro: %.2f\n",mult_rastro);

    fin = fgets(buffer,100,fp);
    if(fin==NULL)
       return(FALSE);
    cargo = sscanf(buffer," %*s %f", &min_visib);
    printf("cargue min_visib: %.2f\n",min_visib);

    fin = fgets(buffer,100,fp);
    if(fin==NULL)
       return(FALSE);
    cargo = sscanf(buffer," %*s %f", &max_visib);
    printf("cargue max_visib: %.2f\n",max_visib);

    fin = fgets(buffer,100,fp);
    if(fin==NULL)
       return(FALSE);
    cargo = sscanf(buffer," %*s %f", &min_prob);
    printf("cargue min_prob: %.2f\n",min_prob);

    fin = fgets(buffer,100,fp);
    if(fin==NULL)
       return(FALSE);
    cargo = sscanf(buffer," %*s %f", &min_rastro);
    printf("cargue min_rastro: %.2f\n",min_rastro);


    return(cargo);

 }


//*************************  cargar parametros ************


void cargar_parametros( char *arch )
{

 FILE *fp;
 int carga_ok;

 fp=fopen(arch,"r");

 if (!fp)
 {
    fprintf(stderr,"ERROR:Archivo %s (parámetros) no encontrado\n",arch);
   exit(-1);
 }

 carga_ok = cargar_vars_globales(fp);
 if (!carga_ok)
 {
    fprintf(stderr,"ERROR:Faltan datos en el archivo %s \n",arch);
    exit(-1);
 }

 fclose(fp);
```

```
}


// *************** Cargar nombres de archivos ******************

void cargar_noms_archivos( char *indice, char nom_arch1[30],
                                char nom_arch2[30], char nom_arch3[30],
                                char nom_arch4[30] )
{
 FILE *fp;
 int i;
 char car_act;

 fprintf(salida,"\nAbriendo archivo Indice...\n");

 fp=fopen(indice,"r");
 if (!fp)
 {
    fprintf(stderr,"ERROR:Archivo %s (indice) no encontrado\n",indice);
   exit(-1);
 }

 fgets(nom_arch1,30,fp);
 fgets(nom_arch2,30,fp);
 fgets(nom_arch3,30,fp);
 fgets(nom_arch4,30,fp);

 fclose(fp);


 for(i=0;i<=30;i++)
 {
    car_act=nom_arch1[i];
    if (car_act=='\n')
         nom_arch1[i]='\0';
 }

 fprintf(salida,"  Archivo 1: %s\n",nom_arch1);

 for(i=0;i<=30;i++)
 {
    car_act=nom_arch2[i];
   if (car_act=='\n')
         nom_arch2[i]='\0';
 }

 fprintf(salida,"  Archivo 2: %s\n",nom_arch2);


 for(i=0;i<=30;i++)
 {
     car_act=nom_arch3[i];
     if (car_act=='\n')
         nom_arch3[i]='\0';
 }

 fprintf(salida,"  Archivo 3: %s\n",nom_arch3);


 for(i=0;i<=30;i++)
 {
     car_act=nom_arch4[i];
     if (car_act=='\n')
         nom_arch4[i]='\0';
```

```
 }

 fprintf(salida,"  Archivo 4: %s\n\n",nom_arch4);

}


//**************** Cargar valores

void cargar_valores(char buffer[100], char carrera[4], char anio[3])
{
    char var1[4];
    char var2[3];

    sscanf(buffer," %*s %s %*s %s", &var1, &var2);
    strcpy(carrera, var1);
    strcpy(anio, var2);
}



// ******************** obtener cantidades ****************************

// ******************** obtener cantidad de ****************************
// **** carreras, años, cursos, grupos, clases de un grupo ************

void obtener_cants_cur(char *arch_cur)
{
    // obtiene:
    // - cantidad de carreras
    // - maxima cant. de años de una carrera
    // -    "     "   "  cursos de un año
    // -    "     "   "  grupos de un curso
    // -    "     "   "  clases de un grupo.

    FILE *fp;
    char buffer[100];

    char identif[60];
    char carrera[4];
    char anio[3];
    char carrera_ant[4];
    char anio_ant[3];
   int existen_cants;
    int c_anios_aux, c_cursos_aux, c_grupos_aux, c_clases_aux;
    long offset;
    char *fin;


    existen_cants = FALSE;
    strcpy(carrera_ant, "  ");
    strcpy(anio_ant, "  ");

    c_anios_aux = c_cursos_aux = c_grupos_aux = c_clases_aux = 0;

   // Inicializacion de variables globales:
    cant_carreras = cant_anios = cant_cursos = cant_grupos = 0;
    cant_clases_gru = 0;


    fp=fopen(arch_cur,"r");

    if (!fp)
    {
        fprintf(stderr,"ERROR:Archivo %s (facultad) no
```

```
                encontrado\n",arch_cur);
˝
    exit(-1);
    }

    offset = ftell(fp);  // guardo posicion del puntero del archivo.
    fin = fgets(buffer,100,fp);
    while (fin!=NULL)
    {
        strcpy(identif,"  ");

        sscanf(buffer," %s", &identif);

        if(cmp_str("car", identif, 3)) // carrera, año, curso.
        {
            cargar_valores(buffer, carrera, anio);
            if(!cmp_str(carrera, carrera_ant, 3))
            {
                strcpy(carrera_ant, carrera);
                cant_carreras++;
                if(c_anios_aux > cant_anios)
                    cant_anios = c_anios_aux;
                c_anios_aux = 0;
                if((c_cursos_aux > cant_cursos))
                    cant_cursos = c_cursos_aux;
                c_cursos_aux = 0;
                strcpy(anio_ant,"  ");
            }


            if(!cmp_str(anio, anio_ant, 2))
            {
                strcpy(anio_ant, anio);
                c_anios_aux++;
                if((c_cursos_aux > cant_cursos))
                cant_cursos = c_cursos_aux;
              c_cursos_aux = 0;
            }

            c_cursos_aux++;
            if((c_cursos_aux > cant_cursos))
                cant_cursos = c_cursos_aux;
            c_grupos_aux = 0;
        } // if cmp_str(car)


        if(cmp_str("gru", identif, 3)) // grupo.
        {
            c_grupos_aux++;
            if((c_grupos_aux > cant_grupos))
                cant_grupos = c_grupos_aux;
            c_clases_aux = 0;
        }


        if(cmp_str("cla", identif, 3)) // clase.
        {
            c_clases_aux++;
            if((c_clases_aux > cant_clases_gru))
                cant_clases_gru = c_clases_aux;
        }


        if(cmp_str("can", identif, 3)) // cantidad.
        {
            existen_cants = TRUE;
```

```
            fseek(fp, offset, SEEK_SET); // vuelvo atras.
            cargar_horas(fp);
        }

         offset = ftell(fp);  // guardo posicion del puntero del archivo.
         fin = fgets(buffer,100,fp);

    } // while

    if(!existen_cants)
    {
        fprintf(stderr,"ERROR: Faltan cantidades de 1/2 horas en el
archivo\n");
        fprintf(stderr,"       de datos de facultad \n");
        exit(-1);
    }

    if((c_clases_aux > cant_clases_gru))
        cant_clases_gru = c_clases_aux;
    if((c_grupos_aux > cant_grupos))
        cant_grupos = c_grupos_aux;
    if((c_cursos_aux > cant_cursos))
        cant_cursos = c_cursos_aux;
    if((c_anios_aux > cant_anios))
        cant_anios = c_anios_aux;

    // Debe sumarse uno pues el vector de clases de un grupo se
    // controla por valor (-1):
    cant_clases_gru++;


    fclose(fp);

}


//************** Obtener cantidad de clases ******************

void obtener_cants_cla(char *arch_cla)
{
    // obtiene la cantidad de clases según el archivo de clases.

    FILE *fp;
    char buffer[100];
    char *fin;
    char identif[4];

   // Inicializacion de variable global:
    cant_clases = 0;

    fp=fopen(arch_cla,"r");

    if (!fp)
    {
        fprintf(stderr,"ERROR:Archivo %s (clases) no encontrado\n",arch_cla);
        exit(-1);
    }

    fin = fgets(buffer,100,fp);
    while (fin!=NULL)
    {
        sscanf(buffer,"%s", &identif);
        if(cmp_str("cla", identif, 3))
        cant_clases++;
        fin = fgets(buffer,100,fp);
    }
```

```
    fclose(fp);

}


//************** Obtener cantidad de salones ******************

void obtener_cants_sal(char *arch_sal)
{
    // obtiene la cantidad de salones según el archivo de salones.

    FILE *fp;
    char buffer[100];
    char *fin;
    char identif[4];

   // Inicializacion de variable global:
    cant_salones = 0;

    fp=fopen(arch_sal,"r");

    if (!fp)
    {
        fprintf(stderr,"ERROR:Archivo %s (salones) no
               encontrado\n",arch_sal);
"
        exit(-1);
"
    }
"

"
    fin = fgets(buffer,100,fp);
"
    while (fin!=NULL)
   {
        sscanf(buffer,"%s", &identif);
        if(cmp_str("sal", identif, 3))
        cant_salones++;
        fin = fgets(buffer,100,fp);
   }


    fclose(fp);

}


//******************** Obtener cantidades ***********************

void obtener_cants(char *arch_cur, char *arch_cla, char *arch_sal)
{
    obtener_cants_cur(arch_cur);

    obtener_cants_cla(arch_cla);

    obtener_cants_sal(arch_sal);

}



//***************** dimensionar estructura de facultad **************
```

```
void dimensionar_estructura_facu( t_carreras_fac &estructura_facu,
                              int d1, int d2, int d3, int d4, int d5)
{
     int i, j, k, l, m;

     fprintf(salida,"\nDimensionando estructuras de memoria...\n");

     estructura_facu=(t_elem_carrera*)calloc(d1,sizeof(t_elem_carrera));
     if(estructura_facu==NULL)
     {
        fprintf(salida,"no se pudo asignar memoria para estructura_facu\n");
        exit(-1);
     }
     for(i=0;i<d1;i++)
     {
        estructura_facu[i].anios_car=
          (t_elem_anio*)calloc(d2,sizeof(t_elem_anio));
        if(estructura_facu[i].anios_car==NULL)
        {
            fprintf(salida,"no se pudo asignar memoria para anios_car\n");
            exit(-1);
        }
        strcpy(estructura_facu[i].id_carrera, " \0");
        for(j=0;j<d2;j++)
        {
          estructura_facu[i].anios_car[j].cursos_anio=
               (t_elem_curso*)calloc(d3,sizeof(t_elem_curso));
          if(estructura_facu[i].anios_car[j].cursos_anio==NULL)
          {
              fprintf(salida,"no se pudo asignar memoria para cursos_anio\n");
              exit(-1);
          }
          strcpy(estructura_facu[i].anios_car[j].id_anio, " \0");

          for(k=0;k<d3;k++)
          {
             estructura_facu[i].anios_car[j].cursos_anio[k].grupos_cur=
                 (t_elem_grupo*)calloc(d4,sizeof(t_elem_grupo));
             if(estructura_facu[i].anios_car[j].
                   cursos_anio[k].grupos_cur==NULL)
             {
                 fprintf(salida,"no se pudo asignar memoria para
                           grupos_cur\n");
˝
                 exit(-1);
˝
             }
˝
             strcpy(estructura_facu[i].anios_car[j].
                      cursos_anio[k].id_curso, " \0");
˝

˝
             for(l=0;l<d4;l++)
˝
             {
˝
                estructura_facu[i].anios_car[j].cursos_anio[k].
                grupos_cur[l].clases_gru = (int *)calloc(d5,sizeof(int));
˝
                if(estructura_facu[i].anios_car[j].cursos_anio[k].
                    grupos_cur[l].clases_gru==NULL)
˝
```

```
                {
                    fprintf(salida,"no se pudo asignar memoria para
                                clases_gru\n");
                    exit(-1);
                }
                strcpy(estructura_facu[i].anios_car[j].cursos_anio[k].
                       grupos_cur[l].id_grupo, " \0");

                for(m=0;m<d5;m++)

                estructura_facu[i].anios_car[j].cursos_anio[k].
                    grupos_cur[l].clases_gru[m]=(-1);
            }

          }
        }
     }
     if( estructura_facu==NULL)
     {
        fprintf(salida,"No se pudo asignar memoria para estructura_facu\n");
        exit(-1);
     }

}



 //************ dimensionar estructuras **********************************

 void dimensionar_estructuras()
 {

    dimensionar_estructura_facu( estructura_facu, cant_carreras, cant_anios,
                         cant_cursos, cant_grupos,cant_clases_gru);


   clases_fac=(clase*)calloc(cant_clases,sizeof(clase)); //dim de clases_fac

   salones_fac=(salon*)calloc(cant_salones,sizeof(salon)); //dim de
                                                            //salones_fac


    v_turnos.tope=0;
 }



//****************** Carga de dimensiones ************************
// Se cargan a partir de los archivos la cantidad de clases, salones,
// carreras, etc.

void carga_de_dimensiones(char *arch_indice)
{
    char arch_cursos[30];
    char arch_clases[30];
    char arch_salones[30];
    char arch_parametros[30];
```

```
    cargar_noms_archivos(arch_indice, arch_cursos, arch_clases,
                         arch_salones, arch_parametros);
˜

˜
    obtener_cants(arch_cursos, arch_clases, arch_salones);
˜

˜
   fprintf(salida, "Dimensiones de las estructuras:\n");
˜
    fprintf(salida, " cant_carr       %i\n",cant_carreras);
    fprintf(salida, " cant_anios      %i\n",cant_anios);
    fprintf(salida, " cant_cursos     %i\n",cant_cursos);
    fprintf(salida, " cant_grupos     %i\n",cant_grupos);
    fprintf(salida, " cant_clases_gru %i\n",cant_clases_gru);
    fprintf(salida, " cant_clases     %i\n",cant_clases);
    fprintf(salida, " cant_salones    %i\n",cant_salones);

}



//*********   Carga de datos   ****************************************


void carga_de_datos(char *arch_indice, matriz3d_str &gru_caranio)
{

    char arch_cursos[30];
    char arch_clases[30];
    char arch_salones[30];
    char arch_parametros[30];

    cargar_noms_archivos(arch_indice, arch_cursos, arch_clases,
                         arch_salones, arch_parametros);
˜

˜
    cant_dias_sem = (int)ceil(cant_hors / cant_horas_dia);

   printf("Cantidad de dias de la semana: %i\n",cant_dias_sem);


    dimensionar_estructuras();

    //*****  se cargan las estructuras que están en la memoria con los datos
   //*****  leidos de los archivos de entrada

    cargar_parametros(arch_parametros);


    cargar_carreras( arch_cursos, estructura_facu, clases_fac, v_turnos,
                       gru_caranio );


    fprintf(salida,"\n****** Matriz de grupos de una carrera y año
                      (gru_caranio): ******\n");
˜
    gru_caranio.mostrar();
˜

˜
    cargar_clases_facu( arch_clases, clases_fac );
```

```
    cargar_salones(arch_salones, salones_fac);
}
```

Estrfacu.hpp

```
¨

¨

¨
class salon
¨
{
¨
 public:
¨
   char id_salon[4];
¨
   int ind_salon;
¨
   int capacidad;
¨
    vectint otras_car;
¨
 public:
¨
    salon( char p_id_salon[4], int p_capacidad,
            vectint p_otras_car );
    void consultar_otras_car(vectint p_otras_car);


};



// vector de ubicaciones de una clase en la facultad:

typedef struct
        {
          char carrera[4];
         char anio[4];
          char curso[5];
          char grupo[5];
          int ind_car;
          int ind_anio;
          int ind_cur;
          int ind_gru;
         }t_elem_ubicacion;

typedef struct
         {
          t_elem_ubicacion vect[20];
          int tope;
         }t_vect_ubicacion;



// clase de la facultad:

class clase
{
  public:
     t_vect_ubicacion v_ubicacion;
    char id_clase[10];
    int cant_estud;
    int importancia;
```

```
    int duracion;
    rangos pref_hors;
    vectint otras_pref;  // son booleanos
    int  es_teorico;



  public:
    clase( char p_id_clase[10] );

     void guardar_datos_clase( int p_cant_estud, int p_importancia,
                               int p_duracion, int p_es_teorico,
                               rangos p_pref_hors,
                               vectint p_otras_pref );



    void guardar_ubicacion(char carrera[], char anio[],
                           char curso[], char grupo[], int p_ind_car,
                           int p_ind_anio, int p_ind_cur, int p_ind_gru);



    int consultar_durac();

    void consultar_otras_pref(vectint p_otras_pref);

    void consultar_ubicacion(t_vect_ubicacion &p_v_ubicacion);
    int consultar_es_teo();
};


// vector de clases de la facultad:

typedef clase *t_clases_fac;


// estructura jerárquica de la facultad:

typedef int *t_clases_gru;


typedef struct
         {
          char id_grupo[5];
          int turno;
          int ctrl_eq_hini;
          t_clases_gru clases_gru;
         }t_elem_grupo;

typedef t_elem_grupo *t_grupos_cur;


typedef struct
         {
          char id_curso[5];
          t_grupos_cur grupos_cur;
         }t_elem_curso;
```

```
typedef t_elem_curso *t_cursos_anio;


typedef struct
         {
          char id_anio[3];
          t_cursos_anio cursos_anio;
         }t_elem_anio;

typedef t_elem_anio *t_anios_car;



˝
typedef struct
˝
         {
˝
          char id_carrera[4];
˝
          t_anios_car anios_car;
˝
         }t_elem_carrera;
˝

˝
typedef t_elem_carrera *t_carreras_fac;



// vector de salones:

typedef salon *t_salones;

void obtener_id_salon(int sal, char p_id_salon[4]);

// turnos:

typedef struct
{
    int turno;
    int h_ini;
   int h_fin;
}elem_turno;

typedef struct
{
    elem_turno vect[10];
    int tope;
}t_vect_turnos;


typedef struct
{
    vector_tope_entero vect[10];
    int tope;
}t_perfiles;
```

Estrfacu.cpp

```
#include "auxiliar.hpp"
˜
#include "matrices.hpp"
˜
#include "estrfacu.hpp"
˜

˜
extern t_salones salones_fac;


salon::salon( char p_id_salon[4], int p_capacidad,
              vectint p_otras_car )
{
  strcpy(id_salon,p_id_salon);
  capacidad=p_capacidad;
  arrcpy2(otras_car, p_otras_car);
}


void salon::consultar_otras_car(vectint p_otras_car)
{
  arrcpy2(p_otras_car, otras_car);
}



clase::clase(char p_id_clase[10])

{
 strcpy(id_clase,p_id_clase);
 v_ubicacion.tope=0;

}


void clase::guardar_ubicacion(char carrera[], char anio[],
                              char curso[], char grupo[], int p_ind_car,
                              int p_ind_anio, int p_ind_cur, int p_ind_gru)

{
  t_elem_ubicacion ubicacion;

  strcpy(v_ubicacion.vect[v_ubicacion.tope].carrera, carrera);
  strcpy(v_ubicacion.vect[v_ubicacion.tope].anio, anio);
  strcpy(v_ubicacion.vect[v_ubicacion.tope].curso, curso);
  strcpy(v_ubicacion.vect[v_ubicacion.tope].grupo, grupo);
  v_ubicacion.vect[v_ubicacion.tope].ind_car=p_ind_car;
  v_ubicacion.vect[v_ubicacion.tope].ind_anio=p_ind_anio;
  v_ubicacion.vect[v_ubicacion.tope].ind_cur=p_ind_cur;
  v_ubicacion.vect[v_ubicacion.tope].ind_gru=p_ind_gru;

  v_ubicacion.tope++;
}



void clase::guardar_datos_clase( int p_cant_estud, int p_importancia,
                     int p_duracion, int p_es_teorico, rangos p_pref_hors,
                            vectint p_otras_pref )
{
 cant_estud=p_cant_estud;
 importancia=p_importancia;
```

```
 duracion=p_duracion;
 arrcpy1(pref_hors,p_pref_hors);
 arrcpy2(otras_pref, p_otras_pref);
 es_teorico=p_es_teorico;


}


int clase::consultar_durac()
{
 return( duracion );
}


void clase::consultar_otras_pref(vectint p_otras_pref)
{
  arrcpy2(p_otras_pref, otras_pref);
}



void clase::consultar_ubicacion(t_vect_ubicacion &p_v_ubicacion)
{
    p_v_ubicacion=v_ubicacion;
}

int clase::consultar_es_teo()
{
  return(es_teorico);
}


void obtener_id_salon(int sal, char p_id_salon[4])
{
    strcpy(p_id_salon, salones_fac[sal].id_salon);
}
```

Hormiga.hpp

```
int obtener_dia(int hor);
int obtener_hora_del_dia(int hor);


//******** Funciones para controlar restricciones. **************

int controlar_lista_tabu(vector_tope lista_tabu, int cla);

int controlar_duracion_clase(matriz2d_indice lista_sh,
                                  int sal, int hor, int cla);

int controlar_clase_dentro_dia(int hor, int cla);

//int controlar_perfiles(t_perfiles perfiles,
   //                      int hor, int cla);

int controlar_superposicion_grupos(mat_horscaranio hors_caranio,
                                        int hor, int cla);

int controlar_clase_en_su_turno(int hor, int cla);

int controlar_separacion_clases_grupo(vector_sh sh_clases, int hor,
                                       int cla);


int cumple_restricciones(vector_tope lista_tabu, matriz2d_indice lista_sh,
                         mat_horscaranio hors_caranio, vector_sh sh_clases,
                            int sal, int hor, int cla);



//******** Funciones para asignar una clase: ***********

void asignar(vector_tope &lista_tabu, matriz2d_indice lista_sh,
                     mat_horscaranio hors_caranio, vector_sh sh_clases,
                     matriz2d asignacion,int sal, int hor, int cla) ;

void asignar_bloque(vector_tope &lista_tabu, matriz2d_indice lista_sh,
                     mat_horscaranio hors_caranio, vector_sh sh_clases,
                     matriz2d asignacion, int hor, int cla);


//******************** Hormiga *************************

class hormiga
{
  protected:
     vector_tope     lista_tabu;
```

```
    matriz2d_indice lista_sh;
    matriz2d        asignacion;
    vector_sh       sh_clases;
    mat_horscaranio hors_caranio;

  public:
     hormiga(int n_salones, int n_horarios, int n_clases);
     void resetear();
    int  consultar_lt(int d);
     void insertar_lt(int d);
    int  llena_lt();
    int  consultar_sh(int d1, int d2);
     void insertar_sh(int d1, int d2);
     int  consultar_asig(int d1, int d2);
     void insertar_asig(int d1, int d2, int elem);
     void copiar_asig(matriz2d mat);

     int  cons_sh_salon(int cla);
    void cons_sh_hors(int cla, int *hor_ini, int *hor_fin);
     void ins_sh_clase(int cla, int salon, int hor_ini, int hor_fin);
     void copiar_sh_clase(vector_sh vec );

     void dejar_rastro(float costo, matriz3d_real mat_rastro);

     friend int obtener_siguiente_sh(matriz2d_indice lista_sh, int *p_sal,
                                     int *p_hor);
~

~
     friend void elegir_clase(vector_tope lista_tabu,
                              matriz2d_indice lista_sh,
                              mat_horscaranio hors_caranio,
                              vector_sh sh_clases,
                              matriz3d_real mat_rastros,
~
                              int sal, int hor, int *p_cla);
~
     friend void obtener_id_clase(int cla, t_string10 id_clase);
~

~
     int   mover(matriz3d_real mat_rastros);
     float calcular_costo_asignacion();

};
```

Hormiga.cpp

```
#include <stdio.h>
˜
#include <stdlib.h>
˜
#include <math.h>
˜
#include <alloc.h>
˜
#include "auxiliar.hpp"
˜
#include "matrices.hpp"
˜
#include "hormiga.hpp"
˜
#include "estrfacu.hpp"

#define RAND rand()/RAND_MAX
#define VACIO -1
#define NO_ENCONTRE -1
#define LIBRE -1


extern float max_costo;
extern float mult_rastro;
extern float min_visib;
extern float max_visib;
extern float min_prob;


extern t_carreras_fac estructura_facu;
//vector  con todas las clases de facultad:
extern t_clases_fac clases_fac;
extern t_vect_turnos v_turnos;
extern t_salones salones_fac;
//extern t_perfiles perfiles;

extern int aux_iter;
extern FILE *salida;
extern FILE *fcostos;
extern FILE *frastros;

extern int cant_hors;
extern int cant_salones;
extern int cant_clases;
extern int cant_grupos;
extern int cant_cursos;
extern int cant_anios;
extern int cant_carreras;
extern int cant_horas_dia;
extern int cant_clases_gru;
extern int cant_dias_sem;

extern float exp_rastro;
extern float exp_visib;

extern float peso_ran_hor;
extern float peso_capac_salon;
extern float peso_otros_req;
extern float peso_desper_salon;
extern float peso_espaciamiento;
extern float peso_adyacencia;
extern float peso_compactibilidad;
```

```
extern float costo_ant ,costo_rh, costo_cap, costo_desp;
extern float costo_oreq, costo_comp, costo_espac,costo_adyac;




int vez=0;
int vez_recta=0;


int obtener_dia(int hor)
{
    div_t x;

    x = div(hor,cant_horas_dia);
   return( x.quot);

}

int obtener_hora_del_dia(int hor)
{
 return( fmod(hor, cant_horas_dia));
}



int obtener_turno(int hor)
{

    int i;
    int h_ini_dia, h_dia;
   int tur_ini, tur_fin;

    h_dia = obtener_hora_del_dia( hor);

    for(i=0;i<v_turnos.tope;i++)
    {
        tur_ini=v_turnos.vect[i].h_ini;
        tur_fin=v_turnos.vect[i].h_fin;
        if((tur_ini <= h_dia)&&(h_dia <= tur_fin))
        return(v_turnos.vect[i].turno);
    }

    return(-1);

}



// *************** hormiga ************************

hormiga::hormiga( int n_salones, int n_horarios, int n_clases)
{
  lista_tabu = vector_tope(n_clases);
  lista_sh = matriz2d_indice(n_salones, n_horarios);
  asignacion = matriz2d(n_salones, n_horarios);
  sh_clases = vector_sh(n_clases);
  hors_caranio = mat_horscaranio(cant_carreras, cant_anios, cant_hors);

}

void hormiga::resetear()
{
```

```
 lista_tabu.resetear();
 lista_sh.resetear();
 asignacion.inicializar();
 sh_clases.inicializar();
 hors_caranio.inicializar();
}



int hormiga::consultar_lt(int d)
{
  int el;
  el=lista_tabu.consultar(d);
  return(el);
}


void hormiga::insertar_lt(int d)
{
  lista_tabu.insertar(d);
}


int hormiga::llena_lt()
{
    return(lista_tabu.llena());
}


int hormiga::consultar_sh(int d1, int d2)
{
  return (lista_sh.consultar(d1, d2));
}


void hormiga::insertar_sh(int d1, int d2)
{
  lista_sh.insertar(d1, d2);
}



int hormiga::consultar_asig(int d1, int d2)
{
    return(asignacion.consultar(d1, d2));
}


void hormiga::insertar_asig(int d1, int d2, int elem)
{
  asignacion.insertar(d1, d2, elem);
}


void hormiga::copiar_asig(matriz2d mat)
{
       int sal, hor;
      int dim1, dim2;

       asignacion.consultar_dims(&dim1, &dim2);

       for(sal=0;sal<dim1;sal++)
           for(hor=0;hor<dim2;hor++)
               mat.insertar(sal, hor, asignacion.consultar(sal, hor));
}
```

```
int hormiga::cons_sh_salon(int cla)
{
 return(sh_clases.consultar_salon(cla));
}


void hormiga::cons_sh_hors(int cla, int *hor_ini, int *hor_fin)
{
 sh_clases.consultar_hors(cla, hor_ini, hor_fin);
}


void hormiga::ins_sh_clase(int cla, int salon, int hor_ini, int hor_fin)
{
 sh_clases.insertar(cla, salon, hor_ini, hor_fin);
}


void hormiga::copiar_sh_clase(vector_sh vec )
{
       int cla, sal, hor_ini, hor_fin;
      int dim1;

       dim1=sh_clases.consultar_dim();

       for(cla=0;cla<dim1;cla++)
      {
           sal=sh_clases.consultar_salon(cla);
           sh_clases.consultar_hors(cla,&hor_ini,&hor_fin);
           vec.insertar(cla, sal, hor_ini, hor_fin);
      }

}



void asignar(vector_tope &lista_tabu, matriz2d_indice lista_sh,
              mat_horscaranio hors_caranio, vector_sh sh_clases,
              matriz2d asignacion, int sal, int hor, int cla)
{

    int dur, h, u;
    t_vect_ubicacion v_ubic;
    int fil, col;


    clases_fac[cla].consultar_ubicacion(v_ubic);
    dur=clases_fac[cla].consultar_durac();

    for(h=hor;h<(hor+dur);h++)
    {
     lista_sh.insertar(sal, h);
     asignacion.insertar(sal, h, cla);



˝
     for(u=0;(u<v_ubic.tope);u++) // para cada ubicacion de la clase
˝
     {
˝
        hors_caranio.insertar(v_ubic.vect[u].ind_car,
                              v_ubic.vect[u].ind_anio, h,
˝
```

```
                                v_ubic.vect[u].grupo);
     }
    }


    sh_clases.insertar(cla, sal, hor, hor+dur-1);

    lista_tabu.insertar(cla);

    return;
}




// ************************* Asignar en Bloque ************************

void asignar_bloque(vector_tope &lista_tabu, matriz2d_indice lista_sh,
                    mat_horscaranio hors_caranio, vector_sh sh_clases,
                    matriz2d asignacion,
                    int hor, int cla)
{
    t_vect_ubicacion v_ubic;
    int *clases_grupo; // alias
   int cla_gru;
   int i_car, i_anio, i_cur, i_gru;
    int hor_dia, h_ini, h_fin, dia, hor_tent, hor_tent_ini, sal_tent;
    int i,j,k;
    rangos prefs;
    int cumple_rest;
    int fil, col, ind;

    int azar;
    vector_tope_entero vect_sals_tent(cant_salones * 10);
    vector_tope_entero vect_hors_tent(cant_salones * 10);

    clases_fac[cla].consultar_ubicacion(v_ubic);

    for(k=0;(k<v_ubic.tope);k++) // para cada ubicacion de la clase
    {
        i_car  = v_ubic.vect[k].ind_car;
        i_anio = v_ubic.vect[k].ind_anio;
        i_cur  = v_ubic.vect[k].ind_cur;
        i_gru  = v_ubic.vect[k].ind_gru;

        // obtener clases del grupo de la clase que quiero asignar:
        clases_grupo =
            estructura_facu[i_car].anios_car[i_anio].cursos_anio[i_cur].
            grupos_cur[i_gru].clases_gru;


        j=0;

        while (clases_grupo[j]!=(-1))   // mientras hayan clases en el grupo

        {

            cla_gru=clases_grupo[j];
            if((cla_gru!=cla)&&(clases_fac[cla_gru].consultar_es_teo()==TRUE))
            // que no sea la clase que estoy controlando
            {
```

```
                sh_clases.consultar_hors(cla_gru, &h_ini, &h_fin);
                if (h_fin == 0) // la clase no fue asignada todavía
                {
                    vect_hors_tent.resetear();
                    vect_sals_tent.resetear();
                    ind=0; //indice para vectores

                    // obetenemos el horario correspondiente al primer dia de
                    // la semana para la clase original:
˝
                     hor_tent_ini = fmod(hor, cant_horas_dia);

                    // obtenemos rangos horarios de preferencia de la clase
                    arrcpy1(prefs, clases_fac[cla_gru].pref_hors);

                    for(dia=0; dia < cant_dias_sem; dia++)
                    {
                        hor_tent = hor_tent_ini + dia*cant_horas_dia;
                        i=0;
                        while(prefs[i].hor_ini != prefs[i].hor_fin)
                    {
                            if ((prefs[i].hor_ini <= hor_tent)
                                  && (prefs[i].hor_fin >= hor_tent ))
                            {
                                for(sal_tent=0; sal_tent < cant_salones;
                                    sal_tent++)
                                {

                                    cumple_rest =
                                    cumple_restricciones(lista_tabu, lista_sh,
˝
                                     hors_caranio, sh_clases, sal_tent, hor_tent,
                                        cla_gru );
˝

˝
                                    if (cumple_rest)
˝
                                    {
˝
                                        vect_hors_tent.insertar(ind,hor_tent);
                                        vect_sals_tent.insertar(ind,sal_tent);
                                        ind++;
                                  }

                                   }
                            }
                        i++;
                    } // while
                    } // for dia

                    if(vect_hors_tent.consultar_tope() > 0)
                {
                        // sorteo una de las asignaciones tentativas:
                        azar=(float)RAND*(vect_hors_tent.consultar_tope() -1 );
                        sal_tent = vect_sals_tent.consultar(azar);
                        hor_tent = vect_hors_tent.consultar(azar);

                        lista_sh.consultar_indices(&fil,&col);

                        asignar(lista_tabu,lista_sh, hors_caranio, sh_clases,
                                  asignacion, sal_tent, hor_tent, cla_gru);

                }
```

```
            } // if no fue asignada
          } // if
         j++;
      } // while clases grupo
    } // for ubicaciones

    vect_hors_tent.destr_vector();
    vect_sals_tent.destr_vector();

}




//********************** dejar rastro ***************************

void hormiga::dejar_rastro(float costo, matriz3d_real mat_rastro)
{
    float rastro;
    int sal, hor, cla;
    int dim1, dim2;

    if(costo>max_costo)
    {
        fprintf(salida,"costo > MAX_COSTO\n");
        exit(-1);
    }


    rastro=(mult_rastro/pow(costo,2));
    asignacion.consultar_dims(&dim1, &dim2);

    for(sal=0;sal<dim1;sal++)
        for(hor=0;hor<dim2;hor++)
        {
            cla = asignacion.consultar(sal, hor);
            if (cla != -1)
                mat_rastro.acumular(sal, hor, cla, rastro);
      }

}



// ********************* mover hormiga ***********************



int obtener_siguiente_sh(matriz2d_indice lista_sh, int *p_sal, int *p_hor)
{
    // busca el primer lugar disponible para asignar una clase en la
    // lista_sh. Si llega al fin de la lista_sh sin encontrar un lugar
    // libre, devuelve TRUE.

    int res;
    int ya_asignado=TRUE;
    int fin_matriz=FALSE;
    while ( (ya_asignado==TRUE) && (fin_matriz==FALSE) )
    {
      if (lista_sh.llena()==TRUE)
         fin_matriz=TRUE;
      else
      {
        lista_sh.consultar_indices(p_sal, p_hor);
```

```
        if (lista_sh.asignado(*p_sal,*p_hor)==TRUE)
        {
            lista_sh.act_indices();
        }
        else
             ya_asignado=FALSE;
      }
    }
    return(fin_matriz);

}



//*********************** Restricciones *********************************


//****** Controlar lista tabu     *****************


int controlar_lista_tabu(vector_tope lista_tabu, int cla)
{

    if ((lista_tabu.consultar(cla))==1)
           return(FALSE);

   return(TRUE);

}



//****** Controlar duracion de la clase*******

int controlar_duracion_clase(matriz2d_indice lista_sh,
                                    int sal, int hor, int cla)
{
   int duracion, ind_hor;

    duracion=clases_fac[cla].consultar_durac();
    ind_hor=hor;
    while ((!lista_sh.consultar(sal,ind_hor))
            &&((ind_hor-hor)<duracion)&&(ind_hor<cant_hors))
        ind_hor++;
    if ( ind_hor-hor!=duracion )
    {
        return(FALSE);
    }

   return(TRUE);
}



//*********** Una clase debe comenzar y terminar dentro del mismo dia *****

int controlar_clase_dentro_dia(int hor, int cla)
{
    int dia_ini, dia_fin;
   int duracion;

    duracion=clases_fac[cla].consultar_durac();
    dia_ini=obtener_dia(hor);
    dia_fin=obtener_dia(hor+duracion-1);
```

```
    if (dia_ini!=dia_fin)
    {
        return(FALSE);
    }

   return(TRUE);
}



//******** Las clases de un mismo perfil no deben superponerse

/*
int controlar_perfiles(t_perfiles perfiles,
                           int hor, int cla)
{

    int h_ini, h_fin;
    int superpone;
    int p,c,c2;
    int dur;


    dur=clases_fac[cla].consultar_durac();
    for(p=0;p<perfiles.tope;p++) // recorro perfiles de la facultad.
    {
        vector_tope_entero clases_perf;
        clases_perf=perfiles.vect[p];
        c=0;
        while (( clases_perf.consultar(c)!=cla)
               &&(c<clases_perf.consultar_tope()))
˜
            c++;
˜
        if( c<clases_perf.consultar_tope())  // la clase esta en el perfil
        {
            c2=0;
            superpone=FALSE;
            while ((c2<clases_perf.consultar_tope()) && (!superpone))
            {
                sh_clases.consultar_hors(clases_perf.consultar(c2),
                            &h_ini, &h_fin);
                if ((h_ini<=hor && h_fin>=hor)||
                    (hor<=h_ini && (hor+dur-1)>=h_ini))
                    superpone=TRUE;
                else
                    c2++;
            }
        }
        if (superpone)
            return(FALSE);

   } // end for

    return(TRUE);
 }
 */


//*********** Controlar superposicion **************

int controlar_superposicion_grupos(mat_horscaranio hors_caranio,
                                          int hor, int cla)
{
    //***** Los cursos de una misma carrera y año no deben superponerse ***
```

```
    // Pero sí pueden admitirse superposiciones entre clases de un mismo
    // curso y distintos grupos (por ej. 2 grupos de practico de An.II que
    // tienen clase a la misma hora en distintos salones.
¨
    // La superposicion se controla a nivel de grupos. Clases que pertenecen
    // a un mismo grupo no deben superponerse, aún cuando pertenezcan a
    // distintos cursos.
¨
¨
¨
    t_vect_ubicacion v_ubic;
¨
    int i_car, i_anio, i_cur, i_gru;
¨
    t_string5 id_gru;
¨
    int pert_ids_grupo;
¨
    int duracion, ind_hor;
¨
    int i;
¨
¨
¨
    duracion = clases_fac[cla].consultar_durac();
    clases_fac[cla].consultar_ubicacion(v_ubic);
    for(i=0;i<v_ubic.tope;i++) // recorro ubicaciones de la clase.
    {
        i_car  = v_ubic.vect[i].ind_car;
        i_anio = v_ubic.vect[i].ind_anio;
        strcpy(id_gru, v_ubic.vect[i].grupo);

        for(ind_hor=hor; ind_hor<(hor+duracion); ind_hor++)
        // recorro horarios a los que quiero asignar la clase
        {
¨
            if (ind_hor==cant_hors) // fin de la matriz de asignacion
¨
            return(FALSE);
¨
            else
¨
            {
¨
                pert_ids_grupo =
                    hors_caranio.pertenece(i_car, i_anio, ind_hor, id_gru);
                if (pert_ids_grupo) // ya se ha asignado una clase con el
                                    // mismo grupo
¨
                {
¨
                    return(FALSE);
¨
                }
¨
            } // end else
¨
        } // end for
¨
    } // end for
¨
```

```
    return(TRUE);
}


//************** Controlar clases en su turno *********************

int controlar_clase_en_su_turno(int hor, int cla)
{
    //Controlar que las clases de un grupo se asignen en el turno
    //indicado por este.

    int turno_grupo, turno_tentativo_fin, turno_tentativo_com;
    t_vect_ubicacion v_ubic;
    int duracion;
    int i_car, i_anio, i_cur, i_gru;
    int i;


    duracion=clases_fac[cla].consultar_durac();

    clases_fac[cla].consultar_ubicacion(v_ubic);
    for(i=0;i<v_ubic.tope;i++) // recorro ubicaciones de la clase.
    {

        i_car  = v_ubic.vect[i].ind_car;
        i_anio = v_ubic.vect[i].ind_anio;
        i_cur  = v_ubic.vect[i].ind_cur;
        i_gru  = v_ubic.vect[i].ind_gru;

        turno_grupo = estructura_facu[i_car].anios_car[i_anio].
                      cursos_anio[i_cur].grupos_cur[i_gru].turno;
        turno_tentativo_com = obtener_turno(hor); //la clase debe comenzar
                                                  // dentro del turno

        turno_tentativo_fin = obtener_turno(hor+duracion-1); //la clase debe
                                                 // terminar dentro del turno


        if ((turno_grupo!=turno_tentativo_com)
               ||(turno_grupo!=turno_tentativo_fin))

        {

            return(FALSE);

        }


    } // for


    return(TRUE);
```

```
}


//************** Controlar separacion de las clases de un grupo ***********

int controlar_separacion_clases_grupo(vector_sh sh_clases, int hor, int cla)
{

    //****   No pueden asignarse 2 clases de un mismo grupo el mismo dia
    // a no ser que una clase sea de teorico y la otra de práctico.

    int *clases_grupo; //es un alias ojo!!
    t_vect_ubicacion v_ubic;
    int dia_asignado, dia_tentativo;
    int cla_gru;
    int es_teo_cla, es_teo_cla_gru;
    int i_car, i_anio, i_cur, i_gru;
    int i, j;
    int h_ini, h_fin;


    es_teo_cla=clases_fac[cla].consultar_es_teo();
    clases_fac[cla].consultar_ubicacion(v_ubic);
    for(i=0;i<v_ubic.tope;i++) // recorro ubicaciones de la clase.
    {

        i_car = v_ubic.vect[i].ind_car;
        i_anio = v_ubic.vect[i].ind_anio;
        i_cur = v_ubic.vect[i].ind_cur;
        i_gru = v_ubic.vect[i].ind_gru;

        clases_grupo = estructura_facu[i_car].anios_car[i_anio].
                        cursos_anio[i_cur].grupos_cur[i_gru].clases_gru;


        j=0;

        while( clases_grupo[j]!=(-1)) // mientras hayan clases en el grupo

        {

            if(clases_grupo[j]!=cla) // que no sea la clase que estoy
                                     //controlando
            {
                cla_gru=clases_grupo[j];
                es_teo_cla_gru=clases_fac[cla_gru].consultar_es_teo();

                if (es_teo_cla_gru==es_teo_cla)
                {
                    sh_clases.consultar_hors(clases_grupo[j], &h_ini, &h_fin);
                    if (h_ini!=h_fin) // si la clase ya fue asignada
                    {
                        dia_asignado=obtener_dia(h_ini);
                        dia_tentativo=obtener_dia(hor);
                        if(dia_asignado==dia_tentativo)
                        {
```

```
                        return(FALSE);
                    }
                }
            }
        }
        j++;
    } // while
} // for ubicaciones

return(TRUE);

}


//******************** cumple restricciones ************************

int cumple_restricciones(vector_tope lista_tabu, matriz2d_indice lista_sh,
                         mat_horscaranio hors_caranio, vector_sh sh_clases,
                         int sal, int hor, int cla)
{
    int cumple;


    cumple = controlar_lista_tabu(lista_tabu, cla);
    if(!cumple)
        return(FALSE);

   cumple = controlar_duracion_clase(lista_sh, sal, hor, cla);
    if(!cumple)
        return(FALSE);

    cumple = controlar_clase_dentro_dia(hor, cla);
    if(!cumple)
        return(FALSE);

    //cumple = controlar_perfiles(perfiles, hor, cla);
    //if(!cumple)
        //return(FALSE);

    cumple = controlar_superposicion_grupos(hors_caranio, hor, cla);
    if(!cumple)
        return(FALSE);


    cumple = controlar_clase_en_su_turno(hor, cla);
    if(!cumple)
        return(FALSE);

    cumple = controlar_separacion_clases_grupo(sh_clases, hor, cla);
   if(!cumple)
        return(FALSE);

   //****   Recursos móviles ( retroproyectores )


    return(TRUE);

}




//*************** Visibilidades *********************************
```

```
//****** Controlar que el salon satisfaga otros requerimientos
//************ version para calculo de visibilidad **************



float controlar_otros_req_vis(int sal, int cla)
{
 float costo;
 int i;
 vectint otras_car, otras_pref;

 costo=0;

 salones_fac[sal].consultar_otras_car(otras_car);
 clases_fac[cla].consultar_otras_pref(otras_pref);
 for(i=0;i<10;i++)
 {
   if ((otras_pref[i]==1) && (otras_car[i]==0))
            costo++;
 }
 return(costo);
}



//************ Controlar desperdicio de la capacidad del salon ******
//************ version para calculo de visibilidad **************



float controlar_desper_salon_vis(int sal, int cla )
{
    float costo;
    float sobran;
    float pct;

    costo=0;


    sobran=salones_fac[sal].capacidad-clases_fac[cla].cant_estud;
    if (sobran >0)
    {
          pct=(sobran/salones_fac[sal].capacidad)*100;
          if (pct >30)
            costo++;

    }
    return(costo);
}



//************ Controlar capacidad del salon suficiente ******
//************ para la clase.                           ******
//************ version para calculo de visibilidad **************


float controlar_capac_salon_vis(int sal, int cla)
{
    float costo;

    costo=0;
    if (clases_fac[cla].cant_estud > salones_fac[sal].capacidad)
```

```
                costo=1;
    return(costo);
}



//************ Controlar rangos horarios de preferencia *********
//************ version para calculo de visibilidad **************

float controlar_rangos_horarios_vis(int hor_ini, int cla)
{
    int dist, dist_min, total_dist=0;
    int duracion, hor_fin;
    rangos prefs;
    int i;

     // consultar duracion de cla y obtener hor_fin:

     duracion = clases_fac[cla].consultar_durac();
     hor_fin = hor_ini + duracion;

     // obtenemos rangos horarios de preferencia de la clase

     arrcpy1(prefs,clases_fac[cla].pref_hors);

     dist_min=cant_hors;

     if ( prefs[0].hor_ini==prefs[0].hor_fin )
        dist_min=0;
     else
     {
      i=0;

      while((i<10) && (!( prefs[i].hor_ini==prefs[i].hor_fin)))
      {
       if ( hor_fin > prefs[i].hor_fin )
           dist= hor_fin-prefs[i].hor_fin;
       else
        if ( hor_ini < prefs[i].hor_ini )
           dist= prefs[i].hor_ini- hor_ini;
        else
           dist=0;
       if ( dist<dist_min )
          dist_min=dist;
       i++;
      }
     }
     total_dist+=dist_min;

    return(total_dist);
}




//*************** Calcular visibilidad *********************

float calcular_visibilidad(int sal, int hor, int cla)
{
    float costo,visib;

    costo = peso_ran_hor * controlar_rangos_horarios_vis(hor, cla);
    costo+= peso_capac_salon * controlar_capac_salon_vis(sal, cla);
    costo+= peso_otros_req * controlar_otros_req_vis( sal, cla);
    costo+= peso_desper_salon * controlar_desper_salon_vis( sal, cla );
```

```
    if (costo > max_costo)
    {
        fprintf(stderr,"Costo muy alto en calculo de visibilidad\n");
        exit(-1);
    }
    if (costo==0)
      visib=max_visib;
    else
      visib=1/costo;
    return(visib);
}




// ********************** Calcular probabilidad ************************


float calcular_prob(int sal, int hor,int cla, matriz3d_real mat_rastros)
{
 //La probabilidad de una clase depende del rastro y la visibilidad.

 float rastro, visib, prob;

 rastro=mat_rastros.consultar(sal, hor, cla);

 visib =calcular_visibilidad(sal, hor, cla);


 // si la visibilidad de una clase es muy baja, se pone en cero
 // para que la clase no sea considerada. De este modo, si no
 // existe ninguna clase atractiva para el (s,h), la hormiga no
 // encontrará clases para asignar, y deberá avanzar al siguiente
 // (s,h).
 // Si todas las visibilidades fueran bajas, pero no cero, de todos
 // modos se elegirá alguna clase, no avanzando el (s,h).

 if (visib < min_visib)
    visib = 0;

 prob = pow(rastro,exp_rastro) * pow(visib,exp_visib);

 return(prob);

}




//************************ elegir clase *******************************

void elegir_clase(vector_tope lista_tabu, matriz2d_indice lista_sh,
                      mat_horscaranio hors_caranio, vector_sh sh_clases,
                      matriz3d_real mat_rastros,
                      int sal, int hor, int *p_cla)
{
    // elige la clase a asignar a la pareja (sal, hor) en base a
    // la visibilidad y los rastros.
    // la visibilidad depende de las restricciones y preferencias
    // locales.

  int j;
  float suma;
  float *recta;
```

```
float *recta_sacum;
float azar;
float prob;
float max_prob;

int elijo;


   recta=(float*)calloc(cant_clases,sizeof(float));

   if (recta==NULL)
   {
        fprintf(salida,"No se pudo asignar memoria para recta \n");
        exit(-1);
   }

   recta_sacum=(float*)calloc(cant_clases,sizeof(float));

   if (recta_sacum==NULL)
   {
        fprintf(salida,"No se pudo asignar memoria para recta_sacum \n");
        exit(-1);
   }


   suma=0;

  max_prob=0;
   for(j=0;j<cant_clases;j++)
   {
      if ( cumple_restricciones(lista_tabu, lista_sh, hors_caranio,
              sh_clases, sal, hor, j))
      {
          prob = calcular_prob(sal, hor, j, mat_rastros);
          //prob depende de rastro y visibilidades.
          if ( prob>max_prob)
            max_prob=prob;

         recta[j] = prob;
      }
      else
      {
          recta[j] = 0;
      }
   }


   // Se construye la recta de probabilidades para sortear una clase.
   // Se normaliza la misma para que contenga valores entre 0 y 1.


   if (max_prob==0)
   {
      (*p_cla)=NO_ENCONTRE;
      free(recta_sacum);
      free(recta);
      return;
  }
   else
   {
     suma=0;
      for(j=0;j<cant_clases;j++)
      {
        recta[j]=recta[j]/max_prob;
        recta_sacum[j]=recta[j];
        suma+=recta[j];
```

```
      }
    }

     // se acumulan los valores de la recta:

    for(j=1;j<cant_clases;j++)
    {
       recta[j]=recta[j]+recta[j-1];
     }

    // genero un numero al azar entre 0 y suma

     azar=(float)RAND*suma;

    *p_cla=0;
    while((azar>recta[*p_cla])||(recta[*p_cla]==0))
        (*p_cla)++;

     // si el valor de la recta (sin acumular) para la clase sorteada
     // es menor que min_prob, la clase se descarta:

     if (recta_sacum[*p_cla]<min_prob)
        (*p_cla=NO_ENCONTRE);

     free(recta_sacum);
     free(recta);

}



void obtener_id_clase(int cla, t_string10 id_clase)
{
    strcpy(id_clase, clases_fac[cla].id_clase);
}




// ********************** mover ***********************************

int hormiga::mover(matriz3d_real mat_rastros)
{
    // devuelve 1 si la hormiga pudo moverse, y
    // devuelve 0 si no pudo mover.

    int sal, hor, cla ,fin_matriz;
    int dur, i, j;
    t_string10 id_clase;
    int obtuve_clase;
    t_vect_ubicacion v_ubic;
    int res;
    int fil,col;


    fin_matriz=FALSE;
    obtuve_clase=FALSE;
    while ((!obtuve_clase)&&(!fin_matriz))
    {
       fin_matriz=obtener_siguiente_sh(lista_sh, &sal, &hor);

       if (fin_matriz==FALSE)
       {
          elegir_clase(lista_tabu, lista_sh, hors_caranio, sh_clases,
                       mat_rastros, sal, hor, &cla);
         if (cla==NO_ENCONTRE)
```

```
        {
           if (lista_sh.llena()==TRUE)
               fin_matriz = TRUE;
           else
           lista_sh.act_indices();
        }
        else
        {
           obtuve_clase=TRUE;
        }

      }
    }


    if (fin_matriz)
    {
     return(0);
    }

    obtener_id_clase(cla, id_clase);

    clases_fac[cla].consultar_ubicacion(v_ubic);
    dur=clases_fac[cla].consultar_durac();

    for(i=hor;i<(hor+dur);i++)
    {
     lista_sh.insertar(sal, i);
     asignacion.insertar(sal, i, cla);
     for(j=0;(j<v_ubic.tope);j++) // para cada ubicacion de la clase
     {
        hors_caranio.insertar(v_ubic.vect[j].ind_car,
               v_ubic.vect[j].ind_anio, i, v_ubic.vect[j].grupo);


     }

    }


    sh_clases.insertar(cla, sal, hor, hor+dur-1);


    lista_tabu.insertar(cla);


    if ( clases_fac[cla].consultar_es_teo()==TRUE)

    {
         asignar_bloque(lista_tabu, lista_sh, hors_caranio,
                       sh_clases, asignacion, hor, cla);
         lista_sh.ajustar_indices( sal, hor+dur-1); // ir al siguiente sh
                              // luego de asignar la ultima 1/2 hora de la
                              // primera clase


    }
```

```
   return(1);
}


//***************** Puedo controlar **********************
// Indica si corresponde
// controlar espaciamiento en dias entre clases de un mismo grupo.

int puedo_controlar(int cant_cla, int dia_min, int dia_max )
{
  int dist;
  int puedo;

  dist=dia_max-dia_min;


  puedo=1;

  if (cant_cla==1)
  {
     puedo=0;
     return(puedo);
  }

  if ((cant_cla==2) &&(dist<2))
  {
       puedo=0;
       return(puedo);
  }

  if ((cant_cla==3) &&(dist<3))
  {
       puedo=0;
       return(puedo);
  }

  if ((cant_cla==4) &&(dist<4))
  {
       puedo=0;
       return(puedo);
  }

  if (cant_cla==5)
  {
       puedo=0;
       return(puedo);
  }

  return( puedo);
}



//***************** evaluo espacimiento ******************

float evaluo_espaciamiento(int cant_cla, int dia_min, int dia_max )
{
  int dist;
  float costo;
```

```
  dist=dia_max-dia_min;

  costo=0;

  if (cant_cla==1)
  {
     costo=0;
     return(costo);
  }
  if ((cant_cla==2) &&(dist<2))
  {
       costo=1;
       return(costo);
  }


  if ((cant_cla==3) &&(dist<3))
  {
       costo=1;
       return(costo);
  }

  if ((cant_cla==4) &&(dist<4))
  {
       costo=1;
       return(costo);
  }

  if (cant_cla==5)
  {
       costo=0;
       return(costo);
  }


  return( costo);
}



//*********  Controlar Espaciamiento *************


float controlar_espaciamiento(vector_sh sh_clase, int car, int anio,
                                    int cur, int gru)
{
 float costo, eval;
 int cla;
 int fin_clases_gru, dia_min, dia_max, dia, cant_cla_gru;
 int h_ini, h_fin, ind_cla, dist;
 rangos prefs;
 int i, min_hini, max_hfin;

    cla=0;
    fin_clases_gru=FALSE;
    dia_min=6;
    dia_max=0;
    cant_cla_gru=0;
   costo = 0;

    // controlo si corresponde exigir espac de acuerdo a los rangos horarios
    min_hini=cant_hors;
    max_hfin=0;

   while(!fin_clases_gru)
```

```
{
    ind_cla= estructura_facu[car].anios_car[anio].cursos_anio[cur].
             grupos_cur[gru].clases_gru[cla];
    if (ind_cla!=(-1))
   {
        cant_cla_gru++;
        arrcpy1(prefs,clases_fac[ind_cla].pref_hors);
        i=0;
        while ((i<10)&&(prefs[i].hor_ini!=prefs[i].hor_fin))
        {
          if (prefs[i].hor_ini<min_hini)
             min_hini=prefs[i].hor_ini;
          if (prefs[i].hor_fin>max_hfin)
              max_hfin=prefs[i].hor_fin;
          i++;
      }

   }
    else
       fin_clases_gru=TRUE;
    cla++;

} // while


if (cant_cla_gru !=0)
{
    dia_min=obtener_dia(min_hini);
    dia_max=obtener_dia(max_hfin);

    if (puedo_controlar(cant_cla_gru,dia_min,dia_max))
    {
        // Obtiene cantidad de clases y primer y ultimo dia de clases en
       // la semana para un grupo:

        cla=0;
       fin_clases_gru=FALSE;
       dia_min=6;
       dia_max=0;
       cant_cla_gru=0;

       while(!fin_clases_gru)
      {
           ind_cla= estructura_facu[car].anios_car[anio].
               cursos_anio[cur].grupos_cur[gru].clases_gru[cla];
           if (ind_cla!=(-1))
           {
               cant_cla_gru++;
               sh_clase.consultar_hors(ind_cla,&h_ini,&h_fin);

               dia=obtener_dia(h_ini);
               if (dia<dia_min)
                   dia_min=dia;
               else
                 if (dia > dia_max)
                   dia_max=dia;
        }
            else
               fin_clases_gru=TRUE;
           cla++;
       } // while
        eval=0;
        eval=evaluo_espaciamiento(cant_cla_gru,dia_min,dia_max);
        costo=costo+eval;
    } // if puedo controlar
}
```

```
   return( costo);
}



//****** Controlar  si el salon satisface otros requerimientos ******


float controlar_otros_req(vector_sh sh_clases)
{
 float costo;
 int sal, cla;
 int i;
 vectint otras_car, otras_pref;

 costo=0;

 for(cla=0;cla<cant_clases;cla++)
 {
    sal=sh_clases.consultar_salon(cla);
    salones_fac[sal].consultar_otras_car(otras_car);
    clases_fac[cla].consultar_otras_pref(otras_pref);
    for(i=0;i<10;i++)
    {
       if ((otras_pref[i]==1) && (otras_car[i]==0))
            costo++;
    }
 }

 return(costo);
}


//************ Controlar desperdicio de la capacidad del salon ******


float controlar_desper_salon(vector_sh sh_clases)
{
   float costo;
   int sal, cla;
   float sobran;
   float pct;

   costo=0;

   for(cla=0; cla<cant_clases; cla++)
   {
          sal = sh_clases.consultar_salon(cla);
          sobran=salones_fac[sal].capacidad-clases_fac[cla].cant_estud;
          if (sobran >0 )
        {
           pct=(sobran/salones_fac[sal].capacidad)*100;

           if (pct >30)
              costo++;
          }

   }

   return(costo);
}


//********* Controlar que el salon tenga capacidad suficiente *********
```

```
float controlar_capac_salon(vector_sh sh_clases)
{
    float costo;
    int sal, cla;

    costo=0;

    for(cla=0; cla<cant_clases; cla++)
    {
           sal = sh_clases.consultar_salon(cla);
           if (clases_fac[cla].cant_estud > salones_fac[sal].capacidad)
               costo++;
    }

    return(costo);
}




//************ Controlar rangos horarios de preferencia *****************


float controlar_rangos_horarios(vector_sh sh_clases)
{
    int dist , dist_min, total_dist=0;
    int hor_ini, hor_fin;
    rangos prefs;
   int cla, i;


    for(cla=0;cla<cant_clases;cla++)
    {
     //*** obtenemos horario que se le asigno a la clase

     sh_clases.consultar_hors(cla,&hor_ini,&hor_fin);

     //*** obtenemos rangos horarios de preferencia de la clase

     arrcpy1(prefs,clases_fac[cla].pref_hors);

     dist_min=cant_hors;
     if ( prefs[0].hor_ini==prefs[0].hor_fin )
        dist_min=0;
     else
     {
      i=0;

      while((i<10) && (!( prefs[i].hor_ini==prefs[i].hor_fin)))
      {
       if ( hor_fin > prefs[i].hor_fin )
            dist= hor_fin-prefs[i].hor_fin;
       else
        if ( hor_ini < prefs[i].hor_ini )
           dist= prefs[i].hor_ini- hor_ini;
        else
           dist=0;
       if ( dist<dist_min )
          dist_min=dist;
       i++;
      }
     }
     total_dist+=dist_min;
```

```
    }
    return(total_dist);
}


int puedo_controlar_igual_hini(int car, int anio, int cur, int gru)
{

return(estructura_facu[car].anios_car[anio].cursos_anio[cur].grupos_cur[gru].
ctrl_eq_hini);
}



//****************** Controlar compactibilidad *********************

float controlar_compactibilidad(matriz2d asignacion)
{
    // cuenta la cantidad de 1/2 horas libres (puentes) para cada
    // salon, dia de la semana y turno.

    float costo;
    int sal, hor, cla;
    int dia_sem, tur, tur_fin, tur_ini;
    int encontre_cla, cant_agujeros;
    int hor_ini_real, hor_fin_real;

    costo=0;

    for(sal=0;sal<cant_salones;sal++)
        for (dia_sem=0; dia_sem<cant_dias_sem; dia_sem++)
            for(tur=0;tur<v_turnos.tope;tur++)
                {
                    encontre_cla = FALSE;
                     cant_agujeros = 0;
                    tur_ini = v_turnos.vect[tur].h_ini;
                    tur_fin = v_turnos.vect[tur].h_fin;
                    hor_ini_real = tur_ini + dia_sem * cant_horas_dia;
                    hor_fin_real = tur_fin + dia_sem * cant_horas_dia;
                    for(hor = hor_ini_real; hor <= hor_fin_real; hor++)
                    {
                    cla = asignacion.consultar(sal,hor);
                        if ( (cla!=(-1))&&(!encontre_cla) )
                        encontre_cla = TRUE;
                        if ( (cla==(-1))&&(encontre_cla) )
                                cant_agujeros++;
                        if ( (cla!=(-1))&&(encontre_cla)&&(cant_agujeros!=0) )
                     {
                            costo+= cant_agujeros;
                            cant_agujeros=0;
                        }
                    }
                }

    return(costo);
}




//************* Controlar adyacencia horaria entre   **************
//************* teoricos y practicos y que se dicten **************
//************* en el mismo salón                    **************

//************* Controlar igual hora de comienzo     **************
```

```
//************* entre los teoricos de un grupo       **************


float controlar_adyacencia_hor_teopra(vector_sh sh_clases,
                                              int car, int anio,
                                              int cur, int gru)
{

       int fin_clases_gru = FALSE;
       int cla = 0;
       int ind_cla, cant_cla_teo, cant_cla_pra;
        int cla_teo, cla_pra, sal_teo, sal_pra, i, j;
        int cla_teo_i, cla_teo_j, h_init_i, h_init_j, h_fint_i, h_fint_j ;
        int adyacentes, ady_sal;
        int h_inip, h_finp, h_init, h_fint, hdia_i, hdia_j;
       vector vec_teo(5);
       vector vec_pra(5);
      float costo=0;

       cant_cla_teo=cant_cla_pra=0;


        while(!fin_clases_gru)
      {
         ind_cla= estructura_facu[car].anios_car[anio].
              cursos_anio[cur].grupos_cur[gru].clases_gru[cla];
˝
         if (ind_cla!=(-1))
˝
           {
˝
                  if(clases_fac[ind_cla].es_teorico)
˝
                  {
˝
                      vec_teo.insertar(cant_cla_teo, ind_cla);
˝
                   cant_cla_teo++;
                  }
                  else
                  {
                      vec_pra.insertar(cant_cla_pra, ind_cla);
                      cant_cla_pra++;
                  }

           }
           else
                fin_clases_gru=TRUE;
           cla++;
       }


      // Adyacencia en horario y salon entre teo y practicos

       if ((cant_cla_teo!=0)&&(cant_cla_pra!=0))
       {
           if (cant_cla_teo>=cant_cla_pra)
              // exijo que los practicos sean adyacentes a algun teo.
              for(i=0;i<cant_cla_pra;i++)
              {
                  adyacentes = ady_sal = FALSE;
                  cla_pra = vec_pra.consultar(i);
                  sh_clases.consultar_hors(cla_pra, &h_inip, &h_finp);
                  sal_pra = sh_clases.consultar_salon(cla_pra);
                  for(j=0;j<cant_cla_teo;j++)
                  {
```

```
                cla_teo = vec_teo.consultar(j);
                sh_clases.consultar_hors(cla_teo, &h_init, &h_fint);
                if(((h_fint+1)==h_inip)||((h_finp+1)==h_init))
                {
                    adyacentes=TRUE;
                    sal_teo = sh_clases.consultar_salon(cla_teo);
                    if(sal_pra==sal_teo)
                        ady_sal=TRUE;
                    break;
                }
            }
            if(!adyacentes)
        {
                costo++;
            }
            else
            {
                if(!ady_sal)
                    costo++;
            }
    }

    else
        // exijo que los teo. sean adyacentes a algun practico.
        for(i=0;i<cant_cla_teo;i++)
        {
        adyacentes=FALSE;
            cla_teo = vec_teo.consultar(i);
            sh_clases.consultar_hors(cla_teo, &h_init, &h_fint);
            sal_teo = sh_clases.consultar_salon(cla_teo);
            for(j=0;j<cant_cla_pra;j++)
            {
                cla_pra = vec_pra.consultar(j);
                sh_clases.consultar_hors(cla_pra, &h_inip, &h_finp);
                if(((h_fint+1)==h_inip)||((h_finp+1)==h_init))
            {
                    adyacentes=TRUE;
                    sal_pra = sh_clases.consultar_salon(cla_pra);
                    if(sal_pra==sal_teo)
                        ady_sal=TRUE;
                    break;
                }
            }
            if(!adyacentes)
             {
                costo++;
            }
            else
        {
                if(!ady_sal)
                    costo++;
            }

        }
}


//*********** Igual hora de comienzo en clases teoricas
//***********       del mismo grupo


if (puedo_controlar_igual_hini(car, anio, cur, gru))
{
    for(i=0;i<cant_cla_teo;i++)
{
        cla_teo_i = vec_teo.consultar(i);
```

```
                sh_clases.consultar_hors(cla_teo_i, &h_init_i, &h_fint_i);

                for(j=i+1;j<cant_cla_teo;j++)
                {

                    cla_teo_j = vec_teo.consultar(j);
                    sh_clases.consultar_hors(cla_teo_j, &h_init_j, &h_fint_j);
                    hdia_i=obtener_hora_del_dia(h_init_i);
                    hdia_j=obtener_hora_del_dia(h_init_j);
                    costo+=abs(hdia_i-hdia_j);
                }
            }

        }


    vec_teo.destr_vector();
    vec_pra.destr_vector();

    return(costo);

}




// ************* Calcular costo asignacion (funcion objetivo) *******
// ************* Aqui se controlan las preferencias

float hormiga::calcular_costo_asignacion()
{

    float costo, costo_esp, costo_ady, costo_total;
    int sal, hor, cla;
    int car, anio, cur, gru;

    costo_ady=0;
    costo_esp=0;
    costo=0;
    costo_total=0;

    float aux_rh, aux_cap, aux_desp, aux_oreq, aux_comp,aux_esp,aux_ady;


    costo = peso_capac_salon * controlar_capac_salon( sh_clases );
    costo_total+= costo;

   aux_cap=costo;
    costo = peso_ran_hor * controlar_rangos_horarios( sh_clases );
    costo_total+= costo;

   aux_rh=costo;
    //if (fmod(aux_iter, 40)==0)
      fprintf(fcostos,"ranhor %.2f  ",costo);

    costo = peso_desper_salon * controlar_desper_salon( sh_clases );
    costo_total+= costo;

   aux_desp=costo;
    //if (fmod(aux_iter, 40)==0)
      fprintf (fcostos,"despsal %.2f  ",costo);

    costo = peso_otros_req * controlar_otros_req( sh_clases );
    costo_total+= costo;
```

```
   aux_oreq=costo;
    //if (fmod(aux_iter, 10)==0)
    //fprintf(fcostos,"otrosre %.2f  ",costo);

    costo = peso_compactibilidad * controlar_compactibilidad(asignacion);
    costo_total+= costo;

   aux_comp=costo;
    //if (fmod(aux_iter, 10)==0)
     fprintf(fcostos,"compact %.2f  ",costo);


   for(car=0;car<cant_carreras;car++)
    for(anio=0;anio<cant_anios;anio++)
      for(cur=0;cur<cant_cursos;cur++)
       for(gru=0;gru<cant_grupos;gru++)
       {
          costo_esp+= peso_espaciamiento *
           controlar_espaciamiento(sh_clases, car, anio, cur, gru);
˜
           costo_ady+= peso_adyacencia *
           controlar_adyacencia_hor_teopra(
                              sh_clases, car, anio, cur, gru);
˜
       }
˜

˜
    costo_total+= costo_esp;
˜

˜
   aux_esp=costo_esp;
˜
    //if (fmod(aux_iter, 10)==0)
˜
        fprintf(fcostos,"espaciam %.2f  ",costo_esp);
˜
    costo_total+= costo_ady;
˜
    //if (fmod(aux_iter, 10)==0)
      fprintf(fcostos,"ady. hor %.2f  ",costo_ady);
   aux_ady=costo_ady;



    if (costo_total<costo_ant)
   {
      costo_rh=aux_rh;
      costo_cap=aux_cap;
      costo_desp=aux_desp;
      costo_oreq=aux_oreq;
      costo_comp=aux_comp;
      costo_espac=aux_esp;
      costo_adyac=aux_ady;
   }


    return (costo_total);
}
```

Matrices.hpp

```
//************* matriz de enteros de 3 dimensiones *************

class matriz3d
{
  protected:
     int ***matriz;
     int dim1, dim2, dim3;

  public:
     matriz3d();
      matriz3d(int d1, int d2, int d3);
     int  consultar(int d1, int d2, int d3);
      void insertar(int d1, int d2, int d3, int elem);
      void inicializar(int valor);
      void acumular(int d1, int d2, int d3, int elem);
     void actualizar(matriz3d mat);
      void mostrar();

  };



//************* matriz de enteros de 2 dimensiones *************

class matriz2d
{
  protected:
     int **matriz;
     int dim1, dim2;

  public:
     matriz2d();
     matriz2d(int d1, int d2);
      int  consultar(int d1, int d2);
      void insertar(int d1, int d2, int elem);
      void inicializar();
      void copiar(matriz2d mat);
      void consultar_dims(int *d1, int *d2);
      void mostrar();
      void mostrar_dia(int dia);
      friend void obtener_id_clase(int cla, t_string10 id_clase);

  };



//************* vector de enteros *************

class vector
{
    protected:
     int *vect;
     int dim;

  public:
     vector();
     vector(int d);
      void destr_vector();
     int  consultar(int d);
      void insertar(int d, int elem);
```

```
      void inicializar();
     int consultar_dim();

  };


//************* vector de booleanos con tope *************

class vector_tope:public vector
{
    protected:
        int tope;

    public:
        vector_tope();
        vector_tope(int d);
        int llena();
        void resetear();
        void insertar(int elem);
        int consultar_tope();

};


//************* vector de enteros con tope *************

class vector_tope_entero:public vector
{
    protected:
        int tope;

    public:
        vector_tope_entero();
        vector_tope_entero(int d);
        int llena();
        void resetear();
        void insertar(int d, int elem);
        int consultar_tope();
};



//************* matriz de booleanos de 2 dimensiones *************

class matriz2d_bool
{
  protected:
     int **matriz  ;
     int dim1, dim2 ;

  public:
     matriz2d_bool();
     matriz2d_bool(int d1, int d2);
    int  consultar(int d1, int d2);
    void insertar(int d1, int d2);
     void inicializar();

  };



//************* matriz de booleanos de 2 dimensiones *************
//************* con índices                          *************

class matriz2d_indice:public matriz2d_bool
```

```
{
    protected:
        int i_fil, i_col;

    public:
        matriz2d_indice();
        matriz2d_indice(int dim1, int dim2);
        int llena();
        void resetear();
        void insertar(int fil, int col);
        void consultar_indices(int *fil, int *col);
        void act_indices();
        void ajustar_indices(int sal, int hor);
        int asignado(int p_sal, int p_hor);
};


//******************* matriz de asignación ******************
//********** matriz de 2 dimensiones cuyos elementos
//********** son strings de largo 10.

class mat_asignacion
{
  protected:
     t_string10 **matriz;
     int dim1, dim2 ;

  public:
     mat_asignacion();
     mat_asignacion(int d1, int d2);
      void consultar(int d1, int d2, t_string10 elem);
      void insertar(int d1, int d2, t_string10 elem);
     void inicializar();

  };



//************* matriz de reales de 3 dimensiones *************

class matriz3d_real
{
  protected:
      float ***matriz;
     int dim1, dim2, dim3;

  public:
     matriz3d_real();
      matriz3d_real(int d1, int d2, int d3);
      float consultar(int d1, int d2, int d3);
      void insertar(int d1, int d2, int d3, float elem);
      void inicializar(float valor);
      void acumular(int d1, int d2, int d3, float elem);
     void actualizar(matriz3d_real mat);
      void mostrar();

};


//********** elemento del vector_sh **************
//***** contiene salón, horario de inicio y horario
//***** de fin al que fue asignada una clase.

class sh_clase
{
     protected:
```

```
       int salon;
       int h_ini;
       int h_fin;

     public:
       sh_clase();
       sh_clase(int sal, int hor_ini, int hor_fin);
       int consultar_sal();
       void consultar_hors(int *hor_ini, int *hor_fin);
};



//******** vector de clases que indica que salón y
//**** horario se asignó a cada una.

class vector_sh
{
    protected:
      sh_clase *vec;
       int dim;

    public:
       vector_sh();
       vector_sh(int d);
       int consultar_dim();
      int consultar_salon(int cla);
       void consultar_hors(int cla, int *hor_ini, int *hor_fin);
       void insertar(int cla ,int sal, int hor_ini, int hor_fin);
       void inicializar();

};


//*********** vector de identificadores de grupos ***************

class vector_ids_grupo
{
  protected:
      t_string5 *vect;
     int tope;
      int dim;

  public:
       vector_ids_grupo();
       vector_ids_grupo(int d);
       void resetear();
       int pertenece(t_string5 elem);
       void insertar(t_string5 elem);
       int consultar_tope();
       void inicializar();
      void mostrar();
};


//************** matriz de grupos de una carrera y año ************
//***** contiene los grupos que se asignaron a un determinado
//***** horario dentro de una carrera y año:
//***** mat_horscaranio(carrera, año, hora) --> vector de grupos

class mat_horscaranio
{
  protected:
     vector_ids_grupo ***matriz;
     int dim1, dim2, dim3;
```

```
  public:
     mat_horscaranio();
     mat_horscaranio(int d1, int d2, int d3);
      int pertenece(int d1, int d2,  int d3, t_string5 elem);
      void insertar(int d1, int d2, int d3, t_string5 elem);
      void inicializar();
     void mostrar();

  };



// elemento de matriz2d_gru (id_clase, salon):

class t_elem_cs
{
  protected:
    t_string10 id_clase;
    char id_salon[4];

  public:
    t_elem_cs();
    t_elem_cs( t_string10 p_id_clase, char p_id_salon[4] );
    void insertar( t_string10 p_id_clase, char p_id_salon[4]);
    void cons_id_clase(t_string10 p_id_clase);
    void cons_id_salon(char p_id_salon[4]);
};



// matriz2d_gru: matriz utilizada para mostrar la asignacion
//               de las clases de un grupo en la semana.
//               sus elementos son parejas (id_clase, salon).

class matriz2d_gru
{
  protected:
     t_elem_cs **matriz;
     int dim1, dim2;

  public:
     matriz2d_gru();
      matriz2d_gru(int d1, int d2);
      void consultar(int d1, int d2, t_elem_cs &elem_cs);
      void insertar(int d1, int d2, t_elem_cs elem_cs);
      void inicializar();
      void consultar_dims(int *d1, int *d2);
      void mostrar();

  };



// matriz3d_str: matriz de 3 dimensiones cuyos elementos son strings
//               de 5 caracteres.

class matriz3d_str
{
  protected:
     t_string5 ***matriz;
     int dim1, dim2, dim3;

  public:
```

```
   matriz3d_str();
    matriz3d_str(int d1, int d2, int d3);
    void consultar(int d1, int d2, int d3, t_string5 elem);
    void insertar(int d1, int d2, int d3, t_string5 elem);
    void inicializar();
    void consultar_dims(int *d1, int *d2, int *d3);
    void mostrar();

};
```

Matrices.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include "auxiliar.hpp"
#include "matrices.hpp"

extern float coef_evap;
extern int cant_horas_dia;
extern int cant_dias_sem;
extern float min_rastro;
extern int cant_cursos;
extern int cant_grupos;

extern FILE *salida;
extern FILE *frastros;



matriz3d::matriz3d()
{
    dim1=dim2=dim3=0;
}


matriz3d::matriz3d(int d1, int d2, int d3)
{
    int i ,j;

    matriz=(int***)calloc(d1,sizeof(int **));
    for(i=0;i<d1;i++)
    {
       matriz[i]=(int**)calloc(d2,sizeof(int*));
       for(j=0;j<d2;j++)
            matriz[i][j]=(int *)calloc(d3,sizeof(int));
     }

    if( matriz==NULL)
    {
      fprintf(stderr,"Error: No se pudo asignar memmoria para
              matriz 3d.\n");
       exit(-1);
     }

    dim1=d1;
    dim2=d2;
    dim3=d3;
}



int  matriz3d::consultar(int d1, int d2, int d3)
{
    return matriz[d1][d2][d3];
}




void matriz3d::insertar(int d1, int d2, int d3, int elem)
{
    matriz[d1][d2][d3]=elem;
```

```
}


void matriz3d::inicializar(int valor)
{
    int i, j, k;

   for(i=0;i<dim1;i++)
      for(j=0;j<dim2;j++)
         for(k=0;k<dim3;k++)
           matriz[i][j][k]=valor;

}


void matriz3d::acumular(int d1, int d2, int d3, int elem)
{
    matriz[d1][d2][d3]+=elem;
}



void matriz3d::mostrar()
{
    int i, j, k;


    for(k=0;k<dim3;k++)
    {
    fprintf(salida,"clase %i\n",k);
        for(i=0;i<dim1;i++)
        {
         for(j=0;j<dim2;j++)
            fprintf(salida,"%i  ",matriz[i][j][k]);
         fprintf(salida,"\n");
        }
    }
}



// ********************** Matriz 2d *********************

matriz2d::matriz2d()
{
    dim1=dim2=0;
}

matriz2d::matriz2d(int d1, int d2)
{
     int i ;

    matriz=(int**)calloc(d1,sizeof(int *));
    for(i=0;i<d1;i++)
     matriz[i]=(int*)calloc(d2,sizeof(int));

    if( matriz==NULL)
    {
      fprintf(stderr,"No se pudo asignar memmoria para matriz\n");
      exit(-1);
    }
    dim1=d1;
    dim2=d2;
}
```

```
int  matriz2d::consultar(int d1, int d2)
{
   return matriz[d1][d2];
}



void matriz2d::insertar(int d1, int d2,  int elem)
{
    matriz[d1][d2]=elem;
}


void matriz2d::inicializar()
{
    int i, j;

    for(i=0;i<dim1;i++)
     for(j=0;j<dim2;j++)
          matriz[i][j]=(-1);

}


void matriz2d::copiar(matriz2d mat)
{
       int sal, hor;

       for(sal=0;sal<dim1;sal++)
           for(hor=0;hor<dim2;hor++)
               matriz[sal][hor]=mat.matriz[sal][hor];
}


void matriz2d::consultar_dims(int *d1, int *d2)
{
    *d1=dim1;
    *d2=dim2;
}


void matriz2d::mostrar_dia(int dia)
{
    int s, h, hd;
    int cla;
    t_string10 id_clase;

    fprintf(salida,"\n****** dia %i ******\n",dia);

       for(s=0;s<dim1;s++)
       {
           for(hd=0; hd<cant_horas_dia; hd++)
               fprintf(salida,"I%03i        ",hd);
         fprintf(salida,"\n");

           for(hd=0; hd<cant_horas_dia; hd++)
       {
               h = dia * cant_horas_dia + hd;
           if (matriz[s][h]<0)
                   fprintf(salida,"nada        ");
               else
               {
                   obtener_id_clase(matriz[s][h], id_clase);
                   fprintf(salida,"%s   ",id_clase);
               }
```

```
            }
            fprintf(salida,"\n");

        }


}


void matriz2d::mostrar()
{
    int i,j;


    for(i=0;i<dim1;i++)
    {
        for(j=0;j<dim2;j++)
            fprintf(salida,"I%3i ",j);
        fprintf(salida,"\n");
        for(j=0;j<dim2;j++)
            fprintf(salida," %3i ",matriz[i][j]);
        fprintf(salida,"\n");
    }


    int dia;

    for(dia=0;dia<cant_dias_sem;dia++)
        mostrar_dia(dia);
}



// ********************** Vector *************************

vector::vector()
{
    dim=0;
}

vector::vector(int d)
{

    vect=(int*)calloc(d,sizeof(int ));

    if( vect==NULL)
    {
      fprintf(stderr,"No se pudo asignar memoria para vector\n");
      exit(-1);
    }
    dim=d;
}



void vector::destr_vector()
{
        free(vect);
}


int  vector::consultar(int d)
{
    return vect[d];
}
```

```
void vector::insertar(int d, int elem)
{
    vect[d]=elem;
}


void vector::inicializar()
{
   int i;

   for(i=0;i<dim;i++)
     vect[i]=0;

}


int vector::consultar_dim()
{
   return (dim);
}



// ********************** Matriz 2d bool *********************

matriz2d_bool::matriz2d_bool()
{
   dim1=dim2=0;
}

matriz2d_bool::matriz2d_bool(int d1, int d2)
{
    int i ;

    matriz=(int**)calloc(d1,sizeof(int *));
    for(i=0;i<d1;i++)
     matriz[i]=(int*)calloc(d2,sizeof(int));

    if( matriz==NULL)
    {
      fprintf(stderr,"No se pudo asignar memmoria para matriz\n");
      exit(-1);
    }
    dim1=d1;
    dim2=d2;
}



int  matriz2d_bool::consultar(int d1, int d2)
{
    return matriz[d1][d2];
}



void matriz2d_bool::insertar(int d1, int d2)
{
    matriz[d1][d2]=TRUE;
}


void matriz2d_bool::inicializar()
{
```

```
    int i, j;

   for(i=0;i<dim1;i++)
    for(j=0;j<dim2;j++)
         matriz[i][j]=FALSE;

}



// ******************** matriz de asignacion ********************



mat_asignacion::mat_asignacion()
{
  dim1=dim2=0;
}


mat_asignacion::mat_asignacion(int d1, int d2)
{
    int i ;

    matriz=(t_string10**)calloc(d1,sizeof(t_string10 *));
    for(i=0;i<d1;i++)
     matriz[i]=(t_string10*)calloc(d2,sizeof(t_string10));

    if( matriz==NULL)
    {
      fprintf(stderr,"No se pudo asignar memoria para matriz asignacion\n");
      exit(-1);
    }
    dim1=d1;
    dim2=d2;
}


void mat_asignacion::consultar(int d1, int d2, t_string10 elem)
{
  strcpy(elem, matriz[d1][d2]);
}



void mat_asignacion::insertar(int d1, int d2, t_string10 elem)
{
  strcpy(matriz[d1][d2], elem);
}


void mat_asignacion::inicializar()
{
    int i, j;

   for(i=0;i<dim1;i++)
    for(j=0;j<dim2;j++)
         strcpy(matriz[i][j]," ");

}


// *************** vector tope ************************

vector_tope::vector_tope()
```

```
{}


vector_tope::vector_tope(int d):vector(d)
{
    tope=0;
}

int vector_tope::llena()
{
    if (tope==dim)
        return(TRUE);
    else
        return(FALSE);
}


void vector_tope::resetear()
{
    tope=0;
   inicializar();
}

void vector_tope::insertar(int elem)
{
    vect[elem]=TRUE;
    tope++;
}


int vector_tope::consultar_tope()
{
    return (tope);
}




// **************** vector tope entero ************************

vector_tope_entero::vector_tope_entero()
{}


vector_tope_entero::vector_tope_entero(int d):vector(d)
{
    tope=0;
}

int vector_tope_entero::llena()
{
    if (tope==dim)
        return(TRUE);
    else
        return(FALSE);
}


void vector_tope_entero::resetear()
{
    tope=0;
   inicializar();
}
```

```
void vector_tope_entero::insertar(int d, int elem)
{
    if (d>=dim)
    {
        printf("ERROR: intento de insertar en vector_tope_entero\n");
        printf("       fuera de sus dimensiones.\n");
        exit(-1);
   }
    vect[d]=elem;
    tope++;
}



int vector_tope_entero::consultar_tope()
{
    return (tope);
}




// **************** matriz 2d indice ************************

matriz2d_indice::matriz2d_indice()
{}


matriz2d_indice::matriz2d_indice(int dim1, int dim2):
                      matriz2d_bool(dim1, dim2)
{
    i_fil=0;
   i_col=0;
}

int matriz2d_indice::llena()
{
    // Se considera llena si el indice de filas esta fuera del rango de la
    // matriz, ya que de lo contrario se pierde la posibilidad de insertar
    // en la ultima fila y columna.
    // Se considera solo el indice de filas dado que la matriz se recorre
    // por filas al actualizar o ajustar los indices.

    if (i_fil==dim1)
        return(TRUE);
    else
        return(FALSE);
}



int matriz2d_indice::asignado(int p_sal, int p_hor)
{
  return( matriz[p_sal][p_hor] );
}



void matriz2d_indice::resetear()
{
    i_fil=0;
   i_col=0;
   inicializar();
}
```

```
void matriz2d_indice::insertar(int fil, int col)
{
  int res;

  if(llena())
  {
       fprintf(stderr,"Error: Se intento insertar en matriz2d_indice y
                  esta llena\n");
       exit(-1);
  }

  matriz[fil][col]=TRUE;

  ajustar_indices(fil,col);
}


void matriz2d_indice::consultar_indices(int *fil, int *col)
{
    *fil = i_fil;
    *col = i_col;
}



void matriz2d_indice::act_indices()
{
    // se recorre la matriz por FILAS.

    if(llena())
    {
        fprintf(stderr,"Error: Se intento act. indices en matriz2d_indice y
                   esta llena\n");
        exit(-1);
    }


    if (i_col<dim2-1)
        i_col++;
    else
    {
        i_col=0;
        i_fil++;
    }

}



void matriz2d_indice::ajustar_indices(int sal, int hor)
{
    // se recorre la matriz por FILAS.

    if(llena())
    {
        fprintf(stderr,"Error: Se intento ajustar indices en matriz2d_indice
                   y esta llena\n");
        exit(-1);
    }


    if (hor<dim2-1)
        i_col=hor+1;
    else
    {
        i_col=0;
```

```
        i_fil=sal+1;
    }

}



//****************** matriz 3d real ******************************

matriz3d_real::matriz3d_real()
{
    dim1=dim2=dim3=0;
}

matriz3d_real::matriz3d_real(int d1, int d2, int d3)
{
     int i ,j;

    matriz=(float ***)calloc(d1,sizeof(float **));
    for(i=0;i<d1;i++)
    {
      matriz[i]=(float **)calloc(d2,sizeof(float *));
      for(j=0;j<d2;j++)
         matriz[i][j]=(float *)calloc(d3,sizeof(float));

    }
    if( matriz==NULL)
    {
      fprintf(stderr,"No se pudo asignar memoria para matriz 3d real\n");
      exit(-1);
    }
    dim1=d1;
    dim2=d2;
    dim3=d3;
}




float matriz3d_real::consultar(int d1, int d2, int d3)
{
    if ((d1<0)||(d2<0)||(d3<0))
    {
       fprintf(stderr,"Indice negativo en matriz3d_real::consultar\n");
       exit(-1);
    }

    return matriz[d1][d2][d3];
}



void matriz3d_real::insertar(int d1, int d2, int d3, float elem)
{
    if ((d1<0)||(d2<0)||(d3<0))
    {
       fprintf(stderr,"Indice negativo en matriz3d_real::insertar\n");
       exit(-1);
    }

    matriz[d1][d2][d3]=elem;
  }

  void matriz3d_real::inicializar(float valor)
  {
    int i, j, k;
```

```
   for(i=0;i<dim1;i++)
      for(j=0;j<dim2;j++)
         for(k=0;k<dim3;k++)
            matriz[i][j][k]=valor;

}


void matriz3d_real::acumular(int d1, int d2, int d3, float elem)
{
    if ((d1<0)||(d2<0)||(d3<0))
    {
        fprintf(stderr,"Indice negativo en matriz3d_real::acumular\n");
        exit(-1);
    }

    matriz[d1][d2][d3]+=elem;
}


void matriz3d_real::actualizar(matriz3d_real mat)
{
    int i, j, k;

   for(i=0;i<dim1;i++)
      for(j=0;j<dim2;j++)
         for(k=0;k<dim3;k++)
        {
            matriz[i][j][k]=(matriz[i][j][k]*coef_evap)+mat.consultar(i,j,k);
            if (matriz[i][j][k] < min_rastro)
                matriz[i][j][k] = min_rastro;
         }
}


void matriz3d_real::mostrar()
{
    int i, j, k;

    for(k=0;k<dim3;k++)
    {
    fprintf(frastros,"clase %i\n",k);
        for(i=0;i<dim1;i++)
        {
         for(j=0;j<dim2;j++)
            fprintf(frastros,"%.2f  ",matriz[i][j][k]);
         fprintf(frastros,"\n");
        }
    }
}



// ******** elemento sh_clase del vector_sh:
// Consta de un salón, un horario inicial y un horario final.

sh_clase::sh_clase()
{
    h_ini = h_fin = salon = 0;
}

sh_clase::sh_clase(int sal, int hor_ini, int hor_fin)
{
    salon = sal;
    h_ini = hor_ini;
```

```
    h_fin = hor_fin;
}

int sh_clase::consultar_sal()
{
    return(salon);
}


void sh_clase::consultar_hors(int *hor_ini, int *hor_fin)
{
    *hor_ini=h_ini;
    *hor_fin=h_fin;
}




//*****************vector sh*****************************

vector_sh::vector_sh()
{
    dim=0;
}


vector_sh::vector_sh(int d)
{
    vec=(sh_clase *)calloc(d,sizeof(sh_clase));
    if( vec==NULL)
    {
        fprintf(stderr,"Error: No se pudo asignar memoria para vector_sh.\n");
       exit(-1);
    }
    dim=d;
}



void vector_sh::inicializar()
{
    int i;
    for(i=0;i<dim;i++)
    {
        insertar(i,0,0,0);
    }
}

int vector_sh::consultar_dim()
{
    return(dim);
}



int vector_sh::consultar_salon(int cla)
{
    return(vec[cla].consultar_sal());
}


void vector_sh::consultar_hors(int cla, int *hor_ini, int *hor_fin)
{
    vec[cla].consultar_hors(hor_ini, hor_fin);
}
```

```
void vector_sh::insertar(int cla ,int sal, int hor_ini, int hor_fin)
{
    vec[cla]=sh_clase(sal, hor_ini, hor_fin);
}


// ***************** vector ids grupo ************************
// vector de identificadores de grupos.

vector_ids_grupo::vector_ids_grupo()
{}


vector_ids_grupo::vector_ids_grupo(int d)
{
    vect=(t_string5 *)calloc(d,sizeof(t_string5));

   dim=d;
    tope=0;
}



int vector_ids_grupo::pertenece( t_string5 elem )
{
  int i;
  int encontre;

  encontre=FALSE;
  i=0;
  while( (i<tope) && (!encontre))
  {
      if (cmp_str(vect[i],elem,5))
          encontre=TRUE;
     else
          i++;
  }

  return(encontre);
}



void vector_ids_grupo::insertar( t_string5 elem)
{
    if (tope==dim)
    {
    fprintf(stderr,"Error: se intento insertar en\n");
        fprintf(stderr,"vector_ids_grupo pero esta lleno\n");
        exit(-1);
   }
    strcpy(vect[tope],elem);
    tope++;
}



int vector_ids_grupo::consultar_tope()
{
    return (tope);
}


void vector_ids_grupo::inicializar()
```

```
{
    tope=0;
}

void vector_ids_grupo::mostrar()
{
    int i;

    for(i=0;i<tope;i++)
        fprintf(salida,"%s  ",vect[i]);
}

// ******************** matriz horscaranio ********************



mat_horscaranio::mat_horscaranio()
{
  dim1=dim2=dim3=0;
}


mat_horscaranio::mat_horscaranio(int d1, int d2, int d3)
{
    int i, j, k;

    matriz=(vector_ids_grupo ***)calloc(d1,sizeof(vector_ids_grupo **));
     for(i=0;i<d1;i++)
     {
      matriz[i]=(vector_ids_grupo **)calloc(d2,sizeof(vector_ids_grupo *));
     for(j=0;j<d2;j++)
        matriz[i][j]=(vector_ids_grupo *)
                          calloc(d3,sizeof(vector_ids_grupo));
    }


    if( matriz==NULL)
    {
       fprintf(stderr,"No se pudo asignar memoria para matriz
hors_caranio\n");
      exit(-1);
    }
    dim1=d1;
     dim2=d2;
     dim3=d3;

     for(i=0;i<dim1;i++)
     for(j=0;j<dim2;j++)
      for(k=0;k<dim3;k++)
          matriz[i][j][k]=vector_ids_grupo(cant_cursos * cant_grupos);

}



int mat_horscaranio::pertenece(int d1, int d2, int d3, t_string5 elem)
{
  // indica si el elemento esta o no en horscaranio.


    if ((d1<0)||(d2<0)||(d3<0))
    {
        fprintf(stderr,"Indice negativo en mat_hors_caranio::consultar\n");
        exit(-1);
    }
```

```
    return(matriz[d1][d2][d3].pertenece(elem));

}


void mat_horscaranio::insertar(int d1, int d2, int d3, t_string5 elem)
{
    vector_ids_grupo p_ids_grupo;

    if ((d1<0)||(d2<0)||(d3<0))
    {
        fprintf(stderr,"Indice negativo en mat_hors_caranio::insertar\n");
        exit(-1);
    }

    matriz[d1][d2][d3].insertar(elem);

}


void mat_horscaranio::inicializar()
{
    // Inicializa todos los vectores de ids_grupos (tope = 0),
   // que son los elementos de mat_horscaranio.

    int i, j, k;

    for(i=0;i<dim1;i++)
     for(j=0;j<dim2;j++)
      for(k=0;k<dim3;k++)
          matriz[i][j][k].inicializar();

}


void mat_horscaranio::mostrar()
{
    int i, j, k;

    for(i=0;i<dim1;i++)
    {
    fprintf(salida,"carrera %i\n",i);
        for(j=0;j<dim2;j++)
        {
         fprintf(salida,"año %i\n",j);
         for(k=0;k<dim3;k++)
         {
            fprintf(salida,"k: %i  ",k);
            matriz[i][j][k].mostrar();
          fprintf(salida,"\n");
        }
         fprintf(salida,"\n");
        }
    }
}




// ********************** elemento de matriz 2d grupos ************
// el elemento consta de un identificador de clase y un salon.

t_elem_cs::t_elem_cs()
{
    strcpy(id_clase," ");
    strcpy(id_salon," ");
```

```
}

t_elem_cs::t_elem_cs( t_string10 p_id_clase, char p_id_salon[4] )
{
    strcpy(id_clase, p_id_clase);
    strcpy(id_salon, p_id_salon);
}

void t_elem_cs::insertar( t_string10 p_id_clase, char p_id_salon[4])
{
    strcpy(id_clase, p_id_clase);
    strcpy(id_salon, p_id_salon);
}

void t_elem_cs::cons_id_clase(t_string10 p_id_clase)
{
    strcpy(p_id_clase, id_clase);
}

void t_elem_cs::cons_id_salon(char p_id_salon[4])
{
    strcpy(p_id_salon, id_salon);
}



// ********************** Matriz 2d grupos *********************
// matriz para visualizar la asignacion por grupos.

matriz2d_gru::matriz2d_gru()
{
    dim1=dim2=0;
}

matriz2d_gru::matriz2d_gru(int d1, int d2)
{
     int i ;

    matriz=(t_elem_cs**)calloc(d1,sizeof(t_elem_cs*));
    for(i=0;i<d1;i++)
       matriz[i]=(t_elem_cs*)calloc(d2,sizeof(t_elem_cs));

    if( matriz==NULL)
    {
      fprintf(stderr,"No se pudo asignar memoria para matriz2d_gru\n");
      exit(-1);
    }
    dim1=d1;
    dim2=d2;
}


void matriz2d_gru::consultar(int d1, int d2, t_elem_cs &elem)
{
     elem = matriz[d1][d2];
}



void matriz2d_gru::insertar(int d1, int d2,  t_elem_cs elem)
{
    matriz[d1][d2] = elem;
}
```

```
void matriz2d_gru::inicializar()
{
    int i, j;
    t_elem_cs elem_cs("         ", "   ");

    for(i=0;i<dim1;i++)
     for(j=0;j<dim2;j++)
        insertar(i, j, elem_cs);

}



void matriz2d_gru::consultar_dims(int *d1, int *d2)
{
    *d1=dim1;
    *d2=dim2;
}



void matriz2d_gru::mostrar()
{
    int i,j;
    t_elem_cs elem_cs;
    t_string10 id_clase;
    char id_salon[4];


        fprintf(salida,"\n     ");
        for(j=0;j<dim2;j++)
            fprintf(salida,"dia %1i       ",j);
        fprintf(salida,"\n");

        for(i=0;i<dim1;i++)
        {

            fprintf(salida,"h%2i  ", i);
            for(j=0;j<dim2;j++)
            {
                consultar(i, j, elem_cs);
                elem_cs.cons_id_clase(id_clase);
                fprintf(salida,"%s   ",id_clase);
            }
            fprintf(salida,"\n     ");

            for(j=0;j<dim2;j++)
            {
                consultar(i, j, elem_cs);
                elem_cs.cons_id_salon(id_salon);
                fprintf(salida,"%s         ",id_salon);
            }
          fprintf(salida,"\n");

        }


}



//****************** matriz 3d str *********************************
// matriz de 3 dimensiones cuyos elementos son strings de 5 caracteres
```

```
matriz3d_str::matriz3d_str()
{
    dim1=dim2=dim3=0;
}

matriz3d_str::matriz3d_str(int d1, int d2, int d3)
{
    int i ,j;

   matriz=(t_string5 ***)calloc(d1,sizeof(t_string5 **));
   for(i=0;i<d1;i++)
   {
      matriz[i]=(t_string5 **)calloc(d2,sizeof(t_string5 *));
      for(j=0;j<d2;j++)
         matriz[i][j]=(t_string5 *)calloc(d3,sizeof(t_string5));

   }
   if( matriz==NULL)
   {
       fprintf(stderr,"No se pudo asignar memoria para matriz 3d str\n");
      exit(-1);
   }
   dim1=d1;
    dim2=d2;
    dim3=d3;
}



void matriz3d_str::consultar(int d1, int d2, int d3, t_string5 elem)
{
    if ((d1<0)||(d2<0)||(d3<0))
   {
       fprintf(stderr,"Indice negativo en matriz3d_str::consultar\n");
       exit(-1);
   }

    strcpy(elem, matriz[d1][d2][d3]);
}



void matriz3d_str::insertar(int d1, int d2, int d3, t_string5 elem)
{
    if ((d1<0)||(d2<0)||(d3<0))
   {
       fprintf(stderr,"Indice negativo en matriz3d_str::insertar\n");
       exit(-1);
   }

   strcpy(matriz[d1][d2][d3],elem);
}


void matriz3d_str::inicializar()
{
    int i, j, k;

   for(i=0;i<dim1;i++)
      for(j=0;j<dim2;j++)
         for(k=0;k<dim3;k++)
            strcpy(matriz[i][j][k]," ");

}
```

```
void matriz3d_str::mostrar()
{
    int i, j, k;


    for(i=0;i<dim1;i++)
    {
        fprintf(salida,"carrera: %2i\n",i);
        for(j=0;j<dim2;j++)
        {
            fprintf(salida,"año: %2i  grupos:  ",j);
            for(k=0;k<dim3;k++)
                fprintf(salida,"%s  ",matriz[i][j][k]);
            fprintf(salida,"\n");
        }
    }

    fprintf(salida,"\n");
}
```

Princip.hpp

```
//******** Fin. Determina si el algorimo debe detenerse ********

int fin(int cant_iter);

// ********** insertar clases de un grupo en la matriz mat_gru *******

void insertar_clases_grupo_en_mat_gru(int car, int anio, int cur, int gru,
                                      vector_sh   sol_sh_clase,
                                      matriz2d_gru &mat_gru);

// ********************** mostrar asignacion por grupos **************

void mostrar_asignacion_por_grupos( vector_sh sol_sh_clase );

// *************** Mostrar resultados *****************

void mostrar_resultados(int encontro_sol, matriz2d solucion,
                        matriz3d_real mat_rastros, int iter_min_costo,
                        vector_sh sol_sh_cla, matriz3d_str gru_caranio);
```

Princip.cpp

```
#include <time.h>
#include <math.h>
#include "auxiliar.hpp"
#include "matrices.hpp"
#include "hormiga.hpp"
#include "estrfacu.hpp"
#include "cargadat.hpp"
#include "princip.hpp"

#define VACIO -1


// Parámetros para la estructura de la facultad:

int cant_clases;
int cant_hors;
int cant_salones;
int cant_clases_gru;
int cant_grupos;
int cant_cursos;
int cant_anios;
int cant_carreras;
int cant_horas_dia;
int cant_dias_sem;

// Parámetros del algoritmo:

int cant_horm;
int max_iter;
float exp_rastro;
float exp_visib;
float coef_evap;
float max_costo;
float mult_rastro;
float min_visib;
float max_visib;
float min_prob;
float min_rastro;

// Pesos para cálculo de preferencias:

float peso_ran_hor;
float peso_capac_salon;
float peso_otros_req;
float peso_desper_salon;
float peso_espaciamiento;
float peso_adyacencia;
float peso_compactibilidad;

// Estructura de la facultad:

t_carreras_fac estructura_facu;
t_vect_turnos v_turnos;
t_salones salones_fac;
t_clases_fac clases_fac; // vector con todas las clases de facultad.
t_perfiles perfiles;


// Archivos para controlar resultados:

FILE *salida;
```

```
FILE *frastros;
FILE *fcostos;

int aux_iter=0; // para debuggear
float costo_ant; // se declara global para debuggear
float costo_rh, costo_cap, costo_desp;     //son para debuggear
float costo_oreq, costo_comp, costo_espac, costo_adyac;  //son para debuggear



//******** Fin. Determina si el algorimo debe detenerse ********

int fin(int cant_iter)
{
    if (cant_iter > max_iter)
        return(TRUE);

    return(FALSE);

}

// ********** insertar clases de un grupo en la matriz mat_gru *******

void insertar_clases_grupo_en_mat_gru(int car, int anio, int cur, int gru,
                                      vector_sh   sol_sh_clase,
                                      matriz2d_gru &mat_gru)
{
    int fin_clases_gru;
    int cla, ind_cla;
    t_string10 id_clase;
   char id_salon[4];
    int sal, hor_ini, hor_fin, dia, h, h_dia;
    t_elem_cs elem_cs(" ", " ");
    t_elem_cs ant_elem_cs(" ", " ");


    fin_clases_gru=FALSE;
    cla=0;
    while(!fin_clases_gru)
   {
        ind_cla= estructura_facu[car].anios_car[anio].
                  cursos_anio[cur].grupos_cur[gru].clases_gru[cla];
        if (ind_cla!=(-1))
        {
            obtener_id_clase(ind_cla, id_clase);
            sal = sol_sh_clase.consultar_salon(ind_cla);
            obtener_id_salon(sal, id_salon);
            elem_cs.insertar(id_clase, id_salon);
            sol_sh_clase.consultar_hors(ind_cla, &hor_ini, &hor_fin);
            dia = obtener_dia(hor_ini);
            for(h=hor_ini; h<=hor_fin; h++)
            {
                h_dia = obtener_hora_del_dia(h);
                mat_gru.consultar(h_dia, dia, ant_elem_cs);
                ant_elem_cs.cons_id_clase(id_clase);

                if(!cmp_str(id_clase,"  ",1))
                {
                    fprintf(stderr,"Error: se intento insertar en
                              mat_gru[%i][%i] y\n", h_dia, dia);
                    fprintf(stderr,"ya estaba ocupado por: %s\n", id_clase);
                    exit(-1);
                }

                mat_gru.insertar(h_dia, dia, elem_cs);
```

```
            }
        }
        else
            fin_clases_gru=TRUE;
        cla++;
    } // while

}



// ********************** mostrar asignacion por grupos **************

void mostrar_asignacion_por_grupos( vector_sh sol_sh_clase,
                                        matriz3d_str gru_caranio )
{
    matriz2d_gru mat_gru(cant_horas_dia, cant_dias_sem);
    int car, anio, cur, gru;

    // variables para gru_caranio:
    int gru_ca; // indice.
    t_string5 id_gru_ca; // identificador de grupo.
    int fin_grupos_ca; // indica fin de los grupos de una carrera-año.

    char id_carrera[4];
    char id_anio[3];
    char id_curso[5];
    char id_grupo[5];


    fprintf(salida,"\n*******************************\n");
    fprintf(salida,"**** asignacion por grupos ****\n");
    fprintf(salida,"*******************************\n");


    // recorro estructura de la facultad y cargo matriz de grupo con
    // las clases de un grupo:

    for(car=0;car<cant_carreras;car++)
    {
        strcpy(id_carrera, estructura_facu[car].id_carrera);
        for(anio=0;anio<cant_anios;anio++)
        {
            strcpy(id_anio, estructura_facu[car].anios_car[anio].id_anio);
            gru_ca = 0;
            fin_grupos_ca = FALSE;
            while(!fin_grupos_ca)
            {
                gru_caranio.consultar(car, anio, gru_ca, id_gru_ca);
                if ( strlen(id_gru_ca)==0 )
                    fin_grupos_ca = TRUE;
                else
                {
                    mat_gru.inicializar();
                    for(cur=0;cur<cant_cursos;cur++)
                        for(gru=0;gru<cant_grupos;gru++)
                        {
                            strcpy(id_grupo, estructura_facu[car].
                                anios_car[anio].cursos_anio[cur].
                                grupos_cur[gru].id_grupo);
                            if (cmp_str(id_grupo, " ", 1))
                                break;
                            if(cmp_str(id_grupo, id_gru_ca, 4))
                             // es uno de los grupos correspondientes al grupo
                             //que estoy mostrando ahora
                                insertar_clases_grupo_en_mat_gru(car, anio, cur,
```

```
                                gru, sol_sh_clase, mat_gru);
                        } // for grupos
                } // else fin_grupos_ca

                // mostrar la planilla de clases de un grupo:
                if(!fin_grupos_ca)
                {
                    fprintf(salida,"\n****** car: %s año: %s gru: %s\n",
                        id_carrera, id_anio, id_gru_ca);
                    mat_gru.mostrar();
                }

              gru_ca++;
            } // while grupos_caranio
          } // for anios
      } // for carreras

}



// *************** Mostrar resultados ****************

void mostrar_resultados(int encontro_sol, matriz2d solucion,
                            matriz3d_real mat_rastros, int iter_min_costo,
                            vector_sh sol_sh_cla, matriz3d_str gru_caranio)
{
    if(encontro_sol)
    {
        fprintf(salida,"\n\n*******************************\n");
        fprintf(salida,"\n  S O L U C I O N   F I N A L \n");
        fprintf(salida,"\n******************************* \n\n");
        fprintf(salida,"\n Matriz de Asignación: \n\n");
        solucion.mostrar();
        fprintf(salida,"\n\n\n C O S T O S : \n\n");
        fprintf(salida,"El costo de la mejor solución es: %.2f
                  \n",costo_ant);
        fprintf(salida,"costo_rh es : %.4f \n", costo_rh);
        fprintf(salida,"costo_cap es : %.4f \n",costo_cap);
        fprintf(salida,"costo_desp es : %.4f \n", costo_desp);
        fprintf(salida,"costo_oreq es : %.4f \n", costo_oreq);
        fprintf(salida,"costo_comp es : %.4f \n",costo_comp);
        fprintf(salida,"costo_espac es : %.4f \n",costo_espac);
        fprintf(salida,"costo_ady es : %.4f \n", costo_adyac);

        fprintf(salida,"Iteracion de la mejor solución : %i
                  \n\n",iter_min_costo);

        mostrar_asignacion_por_grupos(sol_sh_cla, gru_caranio);
    }
    else
    {
        fprintf(salida,"\n*******************************\n");
        fprintf(salida,"*** no se encontro solucion ***\n");
        fprintf(salida,"*******************************\n");

        printf("\n*******************************\n");
        printf("*** no se encontro solucion ***\n");
        printf("*******************************\n");

    }


    fprintf(frastros,"mat. rastros definitiva\n");
      mat_rastros.mostrar();
}
```

```
//*********************************************************************
//**********************   M  A  I  N   ******************************
//*********************************************************************


main( int argc, char *argv[] )
{
    // han declarado como globales los datos de la
    // estrucura de la facultad.

    int m;

    if (argc !=2)
    {
     fprintf(stderr,"\nSintaxis de invocacion:\n");
     fprintf(stderr,"ejecutable.exe nom_archivo_índice.txt\n");
     exit(-1);
    }

    salida = fopen("salida.txt","w");
    frastros = fopen("frastros.txt","w");
    fcostos = fopen("fcostos.txt","w");


    carga_de_dimensiones(argv[1]);


    matriz3d_real mat_rastros(cant_salones,cant_hors,cant_clases);

    // matriz con los grupos pertenecientes a una carrera y año.
    // se utiliza para desplegar la asignacion por grupos:
    matriz3d_str gru_caranio(cant_carreras, cant_anios,
                             cant_cursos * cant_grupos);

    carga_de_datos(argv[1], gru_caranio);

    printf("Termino la carga.\n");
```

```
// ********* Algoritmo principal: **********



matriz3d_real mat_rastros_aux(cant_salones, cant_hors, cant_clases);
matriz2d      solucion(cant_salones, cant_hors);

// para debuggear: solucion_trunc:
matriz2d      solucion_trunc(cant_salones, cant_hors);
vector_sh     sol_sh_cla(cant_clases);

hormiga coquita(cant_salones,cant_hors,cant_clases);

float costo;

//float costo_ant; // se usa global para debuggear

time_t hora;
int cant_iter;
int movio; // indica si la hormiga pudo asignar una clase
            // de lo contrario paso a la siguiente hormiga.

int encontro_sol; // indica si el algoritmo encontro al menos
                  // una solucion.
int encontro_sol_iter; // indica si el algoritmo encontro al
                       // menos una solucion en la iteracion actual.

int i, j, k ;
t_string10 elem, cla;
int iter_min_costo;


costo_ant=RAND_MAX;


coquita.resetear();


srand((unsigned) time(&hora));

cant_iter=0;

mat_rastros.inicializar(min_rastro);

encontro_sol = FALSE;


do
{
    mat_rastros_aux.inicializar(0);
    encontro_sol_iter = FALSE;

    for (i=1;i<=cant_horm;i++)  // para cada hormiga
    {
        do
        {
            movio = coquita.mover(mat_rastros);
            aux_iter++;  // para debuggear.
        }
        while ((!coquita.llena_lt())&&(movio));

        if (movio)
        {
            encontro_sol = TRUE;
            encontro_sol_iter = TRUE;
```

```
            costo = coquita.calcular_costo_asignacion();

            //if (fmod(aux_iter, 40)==0)
            //{
                fprintf(fcostos,"iteracion: %i  ",cant_iter);
                fprintf(fcostos,"solucion: %.2f\n", costo);
            //}

            if (costo < costo_ant)
            {
                coquita.copiar_asig(solucion);
                coquita.copiar_sh_clase(sol_sh_cla);
                costo_ant = costo;
            iter_min_costo = cant_iter;
            }


            // almacenar rastro en matriz temporal:

            coquita.dejar_rastro(costo, mat_rastros_aux);

           //  fprintf(salida,"mat. rastros auxiliar\n");
           //  mat_rastros_aux.mostrar();
        }
        else  // para debuggear
        {
            coquita.copiar_asig(solucion_trunc);
            // fprintf(salida,"**** solucion truncada:\n");
            //  solucion_trunc.mostrar();
        }

        coquita.resetear();  // lista tabu, lista sh, asignacion.

    } // para cada hormiga

    // agregar rastros dejados por las hormigas a la matriz de rastros y
    // actualizarla segun el coeficiente de evaporacion:

    if (encontro_sol_iter)
    {
        mat_rastros.actualizar(mat_rastros_aux);
    }

    if (fmod(cant_iter, 100)==0)
    {
        fprintf(frastros,"mat. rastros definitiva en iter:
                  %i\n",cant_iter);
        mat_rastros.mostrar();
    }


    cant_iter++;
    if (fmod(cant_iter, 10)==0)
        printf("Iteracion : %i\n",cant_iter);


}
 while (!fin(cant_iter));

 mostrar_resultados(encontro_sol, solucion, mat_rastros,
                       iter_min_costo, sol_sh_cla, gru_caranio);

 fclose(frastros);
 fclose(salida);
 fclose(fcostos);
```

```
    printf("Terminé. Andá a leer el archivo \n");
    return 0;
}
```

Cambios en los fuentes del algoritmo para implementar la mejora “mejor hormiga”

Matrices.hpp

```
class matriz3d
{
  protected:
     int ***matriz;
     int dim1, dim2, dim3;

  public:
     matriz3d();
      matriz3d(int d1, int d2, int d3);
     int  consultar(int d1, int d2, int d3);
      void insertar(int d1, int d2, int d3, int elem);
      void inicializar(int valor);
      void acumular(int d1, int d2, int d3, int elem);
     void actualizar(matriz3d mat);
      void mostrar();

  };



class matriz2d
{
  protected:
     int **matriz;
     int dim1, dim2;

  public:
     matriz2d();
     matriz2d(int d1, int d2);
      int  consultar(int d1, int d2);
      void insertar(int d1, int d2, int elem);
      void inicializar();
      void copiar(matriz2d mat);
      void consultar_dims(int *d1, int *d2);
      void mostrar();
      void mostrar_dia(int dia);
      friend void obtener_id_clase(int cla, t_string10 id_clase);

  };

class vector
{
    protected:
      int *vect;
      int dim;

  public:
     vector();
     vector(int d);
      void destr_vector();
     int  consultar(int d);
      void insertar(int d, int elem);
      void inicializar();
```

```
     int consultar_dim();

  };



class vector_tope:public vector
{
    protected:
        int tope;

    public:
        vector_tope();
        vector_tope(int d);
        int llena();
        void resetear();
        void insertar(int elem);
        int consultar_tope();

};



class vector_tope_entero:public vector
{
    protected:
        int tope;

    public:
        vector_tope_entero();
    vector_tope_entero(int d);
        int llena();
        void resetear();
        void insertar(int d, int elem);
        int consultar_tope();
};




class matriz2d_bool
{
  protected:
     int **matriz  ;
     int dim1, dim2 ;

  public:
     matriz2d_bool();
     matriz2d_bool(int d1, int d2);
      int  consultar(int d1, int d2);
      void insertar(int d1, int d2);
     void inicializar();

  };



class matriz2d_indice:public matriz2d_bool
{
    protected:
        int i_fil, i_col;

    public:
        matriz2d_indice();
```

```
       matriz2d_indice(int dim1, int dim2);
       int llena();
       void resetear();
       void insertar(int fil, int col);
       void consultar_indices(int *fil, int *col);
       void act_indices();
       void ajustar_indices(int sal, int hor);
       int asignado(int p_sal, int p_hor);
};



class mat_asignacion
{
  protected:
     t_string10 **matriz;
     int dim1, dim2 ;

  public:
     mat_asignacion();
     mat_asignacion(int d1, int d2);
      void consultar(int d1, int d2, t_string10 elem);
      void insertar(int d1, int d2, t_string10 elem);
     void inicializar();

  };


class matriz3d_real
{
  protected:
      float ***matriz;
     int dim1, dim2, dim3;

  public:
     matriz3d_real();
      matriz3d_real(int d1, int d2, int d3);
      float consultar(int d1, int d2, int d3);
      void insertar(int d1, int d2, int d3, float elem);
      void inicializar(float valor);
      void acumular(int d1, int d2, int d3, float elem);
      void actualizar(matriz3d_real mat);
      void actualizar_iter(matriz2d asignacion, float costo);
      void mostrar();

  };


  class sh_clase
  {
     protected:
        int salon;
        int h_ini;
        int h_fin;

     public:
        sh_clase();
        sh_clase(int sal, int hor_ini, int hor_fin);
        int consultar_sal();
        void consultar_hors(int *hor_ini, int *hor_fin);
  };
```

```
class vector_sh
{
    protected:
       sh_clase *vec;
        int dim;

    public:
        vector_sh();
        vector_sh(int d);
        int consultar_dim();
       int consultar_salon(int cla);
        void consultar_hors(int cla, int *hor_ini, int *hor_fin);
        void insertar(int cla ,int sal, int hor_ini, int hor_fin);
        void inicializar();

};


//*********** vector de identificadores de grupos ***************

class vector_ids_grupo
{
  protected:
        t_string5 *vect;
        int tope;
        int dim;

  public:
        vector_ids_grupo();
        vector_ids_grupo(int d);
        void resetear();
        int pertenece(t_string5 elem);
        void insertar(t_string5 elem);
        int consultar_tope();
        void inicializar();
        void mostrar();
};


//************** matriz de grupos de una carrera y año ************

class mat_horscaranio
{
  protected:
     vector_ids_grupo ***matriz;
     int dim1, dim2, dim3;

  public:
     mat_horscaranio();
     mat_horscaranio(int d1, int d2, int d3);
      int pertenece(int d1, int d2,  int d3, t_string5 elem);
      void insertar(int d1, int d2, int d3, t_string5 elem);
      void inicializar();
     void mostrar();

  };



// elemento de matriz2d_gru (id_clase, salon):

class t_elem_cs
{
  protected:
    t_string10 id_clase;
```

```
    char id_salon[4];

  public:
    t_elem_cs();
    t_elem_cs( t_string10 p_id_clase, char p_id_salon[4] );
    void insertar( t_string10 p_id_clase, char p_id_salon[4]);
    void cons_id_clase(t_string10 p_id_clase);
    void cons_id_salon(char p_id_salon[4]);
};



// matriz2d_gru: matriz utilizada para mostrar la asignacion
//               de las clases de un grupo en la semana.
//               sus elementos son parejas (id_clase, salon).

class matriz2d_gru
{
  protected:
     t_elem_cs **matriz;
     int dim1, dim2;

  public:
     matriz2d_gru();
      matriz2d_gru(int d1, int d2);
      void consultar(int d1, int d2, t_elem_cs &elem_cs);
      void insertar(int d1, int d2, t_elem_cs elem_cs);
      void inicializar();
      void consultar_dims(int *d1, int *d2);
      void mostrar();

  };



// matriz3d_str: matriz de 3 dimensiones cuyos elementos son strings
//               de 5 caracteres.

class matriz3d_str
{
  protected:
     t_string5 ***matriz;
     int dim1, dim2, dim3;

  public:
     matriz3d_str();
      matriz3d_str(int d1, int d2, int d3);
      void consultar(int d1, int d2, int d3, t_string5 elem);
      void insertar(int d1, int d2, int d3, t_string5 elem);
      void inicializar();
      void consultar_dims(int *d1, int *d2, int *d3);
      void mostrar();

  };
```

Matrices.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "auxiliar.hpp"
#include "matrices.hpp"

extern float coef_evap;
extern float max_costo;
extern float mult_rastro;

extern int cant_horas_dia;
extern int cant_dias_sem;
extern float min_rastro;
extern int cant_cursos;
extern int cant_grupos;

extern FILE *salida;
extern FILE *frastros;



matriz3d::matriz3d()
{
    dim1=dim2=dim3=0;
}


matriz3d::matriz3d(int d1, int d2, int d3)
{
    int i ,j;

    matriz=(int***)calloc(d1,sizeof(int **));
    for(i=0;i<d1;i++)
    {
       matriz[i]=(int**)calloc(d2,sizeof(int*));
       for(j=0;j<d2;j++)
        matriz[i][j]=(int *)calloc(d3,sizeof(int));
    }

    if( matriz==NULL)
    {
      fprintf(stderr,"Error: No se pudo asignar memmoria para matriz
              3d.\n");
       exit(-1);
    }

    dim1=d1;
    dim2=d2;
    dim3=d3;
}



int  matriz3d::consultar(int d1, int d2, int d3)
{
    return matriz[d1][d2][d3];
}



void matriz3d::insertar(int d1, int d2, int d3, int elem)
{
```

```
    matriz[d1][d2][d3]=elem;
}


void matriz3d::inicializar(int valor)
{
    int i, j, k;

   for(i=0;i<dim1;i++)
      for(j=0;j<dim2;j++)
         for(k=0;k<dim3;k++)
           matriz[i][j][k]=valor;

}


void matriz3d::acumular(int d1, int d2, int d3, int elem)
{
    matriz[d1][d2][d3]+=elem;
}



void matriz3d::mostrar()
{
    int i, j, k;


    for(k=0;k<dim3;k++)
    {
    fprintf(salida,"clase %i\n",k);
        for(i=0;i<dim1;i++)
        {
         for(j=0;j<dim2;j++)
            fprintf(salida,"%i  ",matriz[i][j][k]);
         fprintf(salida,"\n");
        }
    }
}



// ********************** Matriz 2d **********************

matriz2d::matriz2d()
{
    dim1=dim2=0;
}

matriz2d::matriz2d(int d1, int d2)
{
     int i ;

    matriz=(int**)calloc(d1,sizeof(int *));
    for(i=0;i<d1;i++)
     matriz[i]=(int*)calloc(d2,sizeof(int));

    if( matriz==NULL)
    {
      fprintf(stderr,"No se pudo asignar memmoria para matriz\n");
      exit(-1);
    }
    dim1=d1;
    dim2=d2;
}
```

```
int  matriz2d::consultar(int d1, int d2)
{
   return matriz[d1][d2];
}



void matriz2d::insertar(int d1, int d2,  int elem)
{
    matriz[d1][d2]=elem;
}


void matriz2d::inicializar()
{
    int i, j;

    for(i=0;i<dim1;i++)
     for(j=0;j<dim2;j++)
          matriz[i][j]=(-1);

}


void matriz2d::copiar(matriz2d mat)
{
       int sal, hor;

       for(sal=0;sal<dim1;sal++)
           for(hor=0;hor<dim2;hor++)
           matriz[sal][hor]=mat.matriz[sal][hor];
}


void matriz2d::consultar_dims(int *d1, int *d2)
{
    *d1=dim1;
    *d2=dim2;
}


void matriz2d::mostrar_dia(int dia)
{
    int s, h, hd;
    int cla;
    t_string10 id_clase;

    fprintf(salida,"\n****** dia %i ******\n",dia);

       for(s=0;s<dim1;s++)
       {
           for(hd=0; hd<cant_horas_dia; hd++)
               fprintf(salida,"I%03i        ",hd);
         fprintf(salida,"\n");

           for(hd=0; hd<cant_horas_dia; hd++)
       {
               h = dia * cant_horas_dia + hd;
           if (matriz[s][h]<0)
                   fprintf(salida,"nada        ");
               else
               {
                   obtener_id_clase(matriz[s][h], id_clase);
                   fprintf(salida,"%s   ",id_clase);
```

```
            }
        }
        fprintf(salida,"\n");

    }


}


void matriz2d::mostrar()
{
    int i,j;


    for(i=0;i<dim1;i++)
    {
        for(j=0;j<dim2;j++)
            fprintf(salida,"I%3i ",j);
        fprintf(salida,"\n");
        for(j=0;j<dim2;j++)
            fprintf(salida," %3i ",matriz[i][j]);
        fprintf(salida,"\n");
    }


    int dia;

    for(dia=0;dia<cant_dias_sem;dia++)
        mostrar_dia(dia);
}



// ********************** Vector **************************

vector::vector()
{
    dim=0;
}

vector::vector(int d)
{

    vect=(int*)calloc(d,sizeof(int ));

    if( vect==NULL)
    {
      fprintf(stderr,"No se pudo asignar memoria para vector\n");
      exit(-1);
    }
    dim=d;
}



void vector::destr_vector()
{
        free(vect);
}


int  vector::consultar(int d)
{
    return vect[d];
}
```

```
void vector::insertar(int d, int elem)
{
    vect[d]=elem;
}


void vector::inicializar()
{
   int i;

   for(i=0;i<dim;i++)
     vect[i]=0;

}


int vector::consultar_dim()
{
   return (dim);
}



// ********************** Matriz 2d bool *********************

matriz2d_bool::matriz2d_bool()
{
   dim1=dim2=0;
}

matriz2d_bool::matriz2d_bool(int d1, int d2)
{
    int i ;

    matriz=(int**)calloc(d1,sizeof(int *));
    for(i=0;i<d1;i++)
     matriz[i]=(int*)calloc(d2,sizeof(int));

    if( matriz==NULL)
    {
      fprintf(stderr,"No se pudo asignar memmoria para matriz\n");
      exit(-1);
    }
    dim1=d1;
    dim2=d2;
}



int  matriz2d_bool::consultar(int d1, int d2)
{
    return matriz[d1][d2];
}



void matriz2d_bool::insertar(int d1, int d2)
{
    matriz[d1][d2]=TRUE;
}


void matriz2d_bool::inicializar()
```

```
{
    int i, j;

   for(i=0;i<dim1;i++)
    for(j=0;j<dim2;j++)
         matriz[i][j]=FALSE;

}



// ******************** matriz de asignacion ********************


mat_asignacion::mat_asignacion()
{
  dim1=dim2=0;
}


mat_asignacion::mat_asignacion(int d1, int d2)
{
    int i ;

    matriz=(t_string10**)calloc(d1,sizeof(t_string10 *));
    for(i=0;i<d1;i++)
     matriz[i]=(t_string10*)calloc(d2,sizeof(t_string10));

    if( matriz==NULL)
    {
      fprintf(stderr,"No se pudo asignar memoria para matriz asignacion\n");
      exit(-1);
    }
    dim1=d1;
    dim2=d2;
}


void mat_asignacion::consultar(int d1, int d2, t_string10 elem)
{
  strcpy(elem, matriz[d1][d2]);
}



void mat_asignacion::insertar(int d1, int d2, t_string10 elem)
{
  strcpy(matriz[d1][d2], elem);
}


void mat_asignacion::inicializar()
{
    int i, j;

   for(i=0;i<dim1;i++)
    for(j=0;j<dim2;j++)
         strcpy(matriz[i][j]," ");

}


// *************** vector tope ************************
```

```
vector_tope::vector_tope()
{}


vector_tope::vector_tope(int d):vector(d)
{
    tope=0;
}

int vector_tope::llena()
{
    if (tope==dim)
        return(TRUE);
    else
        return(FALSE);
}


void vector_tope::resetear()
{
    tope=0;
   inicializar();
}

void vector_tope::insertar(int elem)
{
    vect[elem]=TRUE;
    tope++;
}


int vector_tope::consultar_tope()
{
    return (tope);
}




// *************** vector tope entero ************************

vector_tope_entero::vector_tope_entero()
{}


vector_tope_entero::vector_tope_entero(int d):vector(d)
{
    tope=0;
}

int vector_tope_entero::llena()
{
    if (tope==dim)
        return(TRUE);
    else
        return(FALSE);
}


void vector_tope_entero::resetear()
{
    tope=0;
   inicializar();
}
```

```
void vector_tope_entero::insertar(int d, int elem)
{
     vect[d]=elem;
     tope++;
}



int vector_tope_entero::consultar_tope()
{
    return (tope);
}




// **************** matriz 2d indice ************************

matriz2d_indice::matriz2d_indice()
{}


matriz2d_indice::matriz2d_indice(int dim1, int dim2):
                      matriz2d_bool(dim1, dim2)
{
    i_fil=0;
   i_col=0;
}

int matriz2d_indice::llena()
{
    // Se considera llena si el indice de filas esta fuera del rango de la
    //matriz, ya que de lo contrario se pierde la posibilidad de insertar en
    //la última fila y columna.
    // Se considera solo el indice de filas dado que la matriz se recorre
    //por filas al actualizar o ajustar los indices.

    if (i_fil==dim1)
        return(TRUE);
    else
        return(FALSE);
}



int matriz2d_indice::asignado(int p_sal, int p_hor)
{
  return( matriz[p_sal][p_hor] );
}



void matriz2d_indice::resetear()
{
    i_fil=0;
   i_col=0;
   inicializar();
}

void matriz2d_indice::insertar(int fil, int col)
{
  int res;

  if(llena())
  {
```

```
        fprintf(stderr,"Error: Se intento insertar en matriz2d_indice y esta
llena\n");
        exit(-1);
  }

  matriz[fil][col]=TRUE;

  ajustar_indices(fil,col);
}


void matriz2d_indice::consultar_indices(int *fil, int *col)
{
    *fil = i_fil;
    *col = i_col;
}



void matriz2d_indice::act_indices()
{
    // se recorre la matriz por FILAS.

    if(llena())
    {
        fprintf(stderr,"Error: Se intento act. indices en matriz2d_indice y
                esta llena\n");
        exit(-1);
    }


    if (i_col<dim2-1)
        i_col++;
    else
    {
        i_col=0;
        i_fil++;
    }

}



void matriz2d_indice::ajustar_indices(int sal, int hor)
{
    // se recorre la matriz por FILAS.

    if(llena())
    {
        fprintf(stderr,"Error: Se intento ajustar indices en matriz2d_indice
                    y esta llena\n");
        exit(-1);
    }


    if (hor<dim2-1)
        i_col=hor+1;
    else
    {
        i_col=0;
        i_fil=sal+1;
    }

}
```

```
//****************** matriz 3d real ******************************

matriz3d_real::matriz3d_real()
{
    dim1=dim2=dim3=0;
}

matriz3d_real::matriz3d_real(int d1, int d2, int d3)
{
     int i ,j;

    matriz=(float ***)calloc(d1,sizeof(float **));
    for(i=0;i<d1;i++)
    {
      matriz[i]=(float **)calloc(d2,sizeof(float *));
      for(j=0;j<d2;j++)
         matriz[i][j]=(float *)calloc(d3,sizeof(float));

    }
    if( matriz==NULL)
    {
      fprintf(stderr,"No se pudo asignar memoria para matriz 3d real\n");
      exit(-1);
    }
    dim1=d1;
    dim2=d2;
    dim3=d3;
}



float matriz3d_real::consultar(int d1, int d2, int d3)
{
    if ((d1<0)||(d2<0)||(d3<0))
    {
       fprintf(stderr,"Indice negativo en matriz3d_real::consultar\n");
       exit(-1);
    }

    return matriz[d1][d2][d3];
}



void matriz3d_real::insertar(int d1, int d2, int d3, float elem)
{
    if ((d1<0)||(d2<0)||(d3<0))
    {
       fprintf(stderr,"Indice negativo en matriz3d_real::insertar\n");
       exit(-1);
    }

    matriz[d1][d2][d3]=elem;
  }

  void matriz3d_real::inicializar(float valor)
  {
    int i, j, k;

   for(i=0;i<dim1;i++)
      for(j=0;j<dim2;j++)
         for(k=0;k<dim3;k++)
            matriz[i][j][k]=valor;
```

```
}


void matriz3d_real::acumular(int d1, int d2, int d3, float elem)
{
    if ((d1<0)||(d2<0)||(d3<0))
   {
       fprintf(stderr,"Indice negativo en matriz3d_real::acumular\n");
       exit(-1);
   }

    matriz[d1][d2][d3]+=elem;
}


void matriz3d_real::actualizar(matriz3d_real mat)
{
    int i, j, k;

   for(i=0;i<dim1;i++)
      for(j=0;j<dim2;j++)
         for(k=0;k<dim3;k++)
        {
            matriz[i][j][k]=(matriz[i][j][k]*coef_evap)+mat.consultar(i,j,k);
            if (matriz[i][j][k] < min_rastro)
                matriz[i][j][k] = min_rastro;
         }
}



void matriz3d_real::actualizar_iter(matriz2d asignacion, float costo)
{
    float rastro;
    int sal, hor, cla, cla_asig;


    if(costo>max_costo)
   {
       fprintf(salida,"costo > max_costo\n");
       exit(-1);
   }

    // el rastro se calcula aqui ya que no existe una matriz temporal de
    // rastros.

    rastro= mult_rastro/pow(costo, 2);

    fprintf(frastros,"El rastro de la hormiga es %.2f \n",rastro);

    for(sal=0;sal<dim1;sal++)
        for(hor=0;hor<dim2;hor++)
        {
            cla_asig = asignacion.consultar(sal, hor);
            for(cla=0;cla<dim3;cla++)
            {
                if (cla == cla_asig)
                {
                    matriz[sal][hor][cla] =
                        (matriz[sal][hor][cla]*coef_evap)+rastro;
                }
                else
                    matriz[sal][hor][cla] = matriz[sal][hor][cla]*coef_evap;
                if (matriz[sal][hor][cla] < min_rastro)
                    matriz[sal][hor][cla] = min_rastro;
```

```
         }
      }

}



void matriz3d_real::mostrar()
{
    int i, j, k;

    for(k=0;k<dim3;k++)
   {
    fprintf(frastros,"clase %i\n",k);
        for(i=0;i<dim1;i++)
        {
         for(j=0;j<dim2;j++)
            fprintf(frastros,"%.2f  ",matriz[i][j][k]);
         fprintf(frastros,"\n");
        }
   }
}



// ******** elemento sh_clase del vector_sh:
// Consta de un salón, un horario inicial y un horario final.

sh_clase::sh_clase()
{
    h_ini = h_fin = salon = 0;
}

sh_clase::sh_clase(int sal, int hor_ini, int hor_fin)
{
    salon = sal;
    h_ini = hor_ini;
    h_fin = hor_fin;
}

int sh_clase::consultar_sal()
{
    return(salon);
}


void sh_clase::consultar_hors(int *hor_ini, int *hor_fin)
{
    *hor_ini=h_ini;
    *hor_fin=h_fin;
}


//*****************vector sh*****************************

vector_sh::vector_sh()
{
    dim=0;
}


vector_sh::vector_sh(int d)
{
    vec=(sh_clase *)calloc(d,sizeof(sh_clase));
    if( vec==NULL)
```

```
    {
        fprintf(stderr,"Error: No se pudo asignar memoria para
                    vector_sh.\n");
       exit(-1);
    }
    dim=d;
}



void vector_sh::inicializar()
{
    int i;
    for(i=0;i<dim;i++)
    {
        insertar(i,0,0,0);
    }
}

int vector_sh::consultar_dim()
{
    return(dim);
}



int vector_sh::consultar_salon(int cla)
{
    return(vec[cla].consultar_sal());
}


void vector_sh::consultar_hors(int cla, int *hor_ini, int *hor_fin)
{
    vec[cla].consultar_hors(hor_ini, hor_fin);
}


void vector_sh::insertar(int cla ,int sal, int hor_ini, int hor_fin)
{
    vec[cla]=sh_clase(sal, hor_ini, hor_fin);
}


// ***************** vector ids grupo ************************
// vector de identificadores de grupos.

vector_ids_grupo::vector_ids_grupo()
{}


vector_ids_grupo::vector_ids_grupo(int d)
{
    vect=(t_string5 *)calloc(d,sizeof(t_string5));

   dim=d;
    tope=0;
}



int vector_ids_grupo::pertenece( t_string5 elem )
{
  int i;
  int encontre;
```

```
  encontre=FALSE;
  i=0;
  while( (i<tope) && (!encontre))
  {
      if (cmp_str(vect[i],elem,5))
          encontre=TRUE;
     else
          i++;
  }

  return(encontre);
}



void vector_ids_grupo::insertar( t_string5 elem)
{
    if (tope==dim)
    {
        fprintf(stderr,"Error: se intento insertar en\n");
        fprintf(stderr,"vector_ids_grupo pero esta lleno\n");
        exit(-1);
   }
    strcpy(vect[tope],elem);
    tope++;
}



int vector_ids_grupo::consultar_tope()
{
    return (tope);
}


void vector_ids_grupo::inicializar()
{
    tope=0;
}

void vector_ids_grupo::mostrar()
{
    int i;

    for(i=0;i<tope;i++)
        fprintf(salida,"%s  ",vect[i]);
}

// ******************** matriz horscaranio ********************



mat_horscaranio::mat_horscaranio()
{
  dim1=dim2=dim3=0;
}


mat_horscaranio::mat_horscaranio(int d1, int d2, int d3)
{
    int i, j, k;

    matriz=(vector_ids_grupo ***)calloc(d1,sizeof(vector_ids_grupo **));
     for(i=0;i<d1;i++)
     {
      matriz[i]=(vector_ids_grupo **)calloc(d2,sizeof(vector_ids_grupo *));
```

```
     for(j=0;j<d2;j++)
        matriz[i][j] = (vector_ids_grupo *)calloc
                                  (d3,sizeof(vector_ids_grupo));
    }



    if( matriz==NULL)
    {
       fprintf(stderr,"No se pudo asignar memoria para matriz
              hors_caranio\n");
      exit(-1);
    }
    dim1=d1;
     dim2=d2;
     dim3=d3;

     for(i=0;i<dim1;i++)
     for(j=0;j<dim2;j++)
      for(k=0;k<dim3;k++)
          matriz[i][j][k]=vector_ids_grupo(cant_cursos * cant_grupos);

}



int mat_horscaranio::pertenece(int d1, int d2, int d3, t_string5 elem)
{
  // indica si el elemento esta o no en horscaranio.


    if ((d1<0)||(d2<0)||(d3<0))
    {
        fprintf(stderr,"Indice negativo en mat_hors_caranio::consultar\n");
        exit(-1);
    }

    return(matriz[d1][d2][d3].pertenece(elem));

}


void mat_horscaranio::insertar(int d1, int d2, int d3, t_string5 elem)
{
    vector_ids_grupo p_ids_grupo;

    if ((d1<0)||(d2<0)||(d3<0))
    {
        fprintf(stderr,"Indice negativo en mat_hors_caranio::insertar\n");
        exit(-1);
    }

    matriz[d1][d2][d3].insertar(elem);

}


void mat_horscaranio::inicializar()
{
    // Inicializa todos los vectores de ids_grupos (tope = 0),
   // que son los elementos de mat_horscaranio.

    int i, j, k;

    for(i=0;i<dim1;i++)
     for(j=0;j<dim2;j++)
```

```
      for(k=0;k<dim3;k++)
         matriz[i][j][k].inicializar();

}


void mat_horscaranio::mostrar()
{
    int i, j, k;

    for(i=0;i<dim1;i++)
   {
    fprintf(salida,"carrera %i\n",i);
        for(j=0;j<dim2;j++)
        {
         fprintf(salida,"año %i\n",j);
         for(k=0;k<dim3;k++)
         {
            fprintf(salida,"k: %i  ",k);
            matriz[i][j][k].mostrar();
          fprintf(salida,"\n");
        }
         fprintf(salida,"\n");
        }
   }
}



// ********************** elemento de matriz 2d grupos ************
// el elemento consta de un identificador de clase y un salon.

t_elem_cs::t_elem_cs()
{
    strcpy(id_clase," ");
    strcpy(id_salon," ");
}

t_elem_cs::t_elem_cs( t_string10 p_id_clase, char p_id_salon[4] )
{
    strcpy(id_clase, p_id_clase);
    strcpy(id_salon, p_id_salon);
}

void t_elem_cs::insertar( t_string10 p_id_clase, char p_id_salon[4])
{
    strcpy(id_clase, p_id_clase);
    strcpy(id_salon, p_id_salon);
}

void t_elem_cs::cons_id_clase(t_string10 p_id_clase)
{
    strcpy(p_id_clase, id_clase);
}

void t_elem_cs::cons_id_salon(char p_id_salon[4])
{
    strcpy(p_id_salon, id_salon);
}



// ********************** Matriz 2d grupos ********************
// matriz para visualizar la asignacion por grupos.
```

```
matriz2d_gru::matriz2d_gru()
{
    dim1=dim2=0;
}

matriz2d_gru::matriz2d_gru(int d1, int d2)
{
     int i ;

    matriz=(t_elem_cs**)calloc(d1,sizeof(t_elem_cs*));
    for(i=0;i<d1;i++)
       matriz[i]=(t_elem_cs*)calloc(d2,sizeof(t_elem_cs));

    if( matriz==NULL)
    {
      fprintf(stderr,"No se pudo asignar memoria para matriz2d_gru\n");
      exit(-1);
    }
    dim1=d1;
    dim2=d2;
}


void matriz2d_gru::consultar(int d1, int d2, t_elem_cs &elem)
{
     elem = matriz[d1][d2];
}



void matriz2d_gru::insertar(int d1, int d2,  t_elem_cs elem)
{
    matriz[d1][d2] = elem;
}



void matriz2d_gru::inicializar()
{
    int i, j;
    t_elem_cs elem_cs("         ", "   ");

    for(i=0;i<dim1;i++)
     for(j=0;j<dim2;j++)
        insertar(i, j, elem_cs);

}



void matriz2d_gru::consultar_dims(int *d1, int *d2)
{
    *d1=dim1;
    *d2=dim2;
}



void matriz2d_gru::mostrar()
{
    int i,j;
    t_elem_cs elem_cs;
    t_string10 id_clase;
    char id_salon[4];
```

```
        fprintf(salida,"\n     ");
        for(j=0;j<dim2;j++)
            fprintf(salida,"dia %1i       ",j);
        fprintf(salida,"\n");

        for(i=0;i<dim1;i++)
        {

            fprintf(salida,"h%2i  ", i);
            for(j=0;j<dim2;j++)
            {
                consultar(i, j, elem_cs);
                elem_cs.cons_id_clase(id_clase);
                fprintf(salida,"%s   ",id_clase);
            }
            fprintf(salida,"\n     ");

            for(j=0;j<dim2;j++)
            {
                consultar(i, j, elem_cs);
                elem_cs.cons_id_salon(id_salon);
                fprintf(salida,"%s         ",id_salon);
            }
          fprintf(salida,"\n");

        }


}



//****************** matriz 3d str **********************************
// matriz de 3 dimensiones cuyos elementos son strings de 5 caracteres


matriz3d_str::matriz3d_str()
{
    dim1=dim2=dim3=0;
}

matriz3d_str::matriz3d_str(int d1, int d2, int d3)
{
    int i ,j;

   matriz=(t_string5 ***)calloc(d1,sizeof(t_string5 **));
   for(i=0;i<d1;i++)
   {
      matriz[i]=(t_string5 **)calloc(d2,sizeof(t_string5 *));
      for(j=0;j<d2;j++)
         matriz[i][j]=(t_string5 *)calloc(d3,sizeof(t_string5));

   }
   if( matriz==NULL)
   {
       fprintf(stderr,"No se pudo asignar memoria para matriz 3d str\n");
      exit(-1);
   }
   dim1=d1;
    dim2=d2;
    dim3=d3;
}
```

```
void matriz3d_str::consultar(int d1, int d2, int d3, t_string5 elem)
{
    if ((d1<0)||(d2<0)||(d3<0))
   {
       fprintf(stderr,"Indice negativo en matriz3d_str::consultar\n");
       exit(-1);
   }

    strcpy(elem, matriz[d1][d2][d3]);
}



void matriz3d_str::insertar(int d1, int d2, int d3, t_string5 elem)
{
    if ((d1<0)||(d2<0)||(d3<0))
   {
       fprintf(stderr,"Indice negativo en matriz3d_str::insertar\n");
       exit(-1);
   }

   strcpy(matriz[d1][d2][d3],elem);
}


void matriz3d_str::inicializar()
{
    int i, j, k;

   for(i=0;i<dim1;i++)
      for(j=0;j<dim2;j++)
         for(k=0;k<dim3;k++)
            strcpy(matriz[i][j][k]," ");

}


void matriz3d_str::mostrar()
{
    int i, j, k;


    for(i=0;i<dim1;i++)
    {
        fprintf(salida,"carrera: %2i\n",i);
        for(j=0;j<dim2;j++)
        {
            fprintf(salida,"año: %2i  grupos:  ",j);
            for(k=0;k<dim3;k++)
                fprintf(salida,"%s  ",matriz[i][j][k]);
            fprintf(salida,"\n");
        }
    }

    fprintf(salida,"\n");
}
```

Principal.cpp

```
#include <time.h>
#include <math.h>
#include "auxiliar.hpp"
#include "matrices.hpp"
#include "hormiga.hpp"
#include "estrfacu.hpp"
#include "cargadat.hpp"
#include "princip.hpp"

#define VACIO -1


// Parámetros para la estructura de la facultad:

int cant_clases;
int cant_hors;
int cant_salones;
int cant_clases_gru;
int cant_grupos;
int cant_cursos;
int cant_anios;
int cant_carreras;
int cant_horas_dia;
int cant_dias_sem;

// Parámetros del algoritmo:

int cant_horm;
int max_iter;
float exp_rastro;
float exp_visib;
float coef_evap;
float max_costo;
float mult_rastro;
float min_visib;
float max_visib;
float min_prob;
float min_rastro;

// Pesos para cálculo de preferencias:

float peso_ran_hor;
float peso_capac_salon;
float peso_otros_req;
float peso_desper_salon;
float peso_espaciamiento;
float peso_adyacencia;
float peso_compactibilidad;

// Estructura de la facultad:

t_carreras_fac estructura_facu;
t_vect_turnos v_turnos;
t_salones salones_fac;
t_clases_fac clases_fac; // vector con todas las clases de facultad.
t_perfiles perfiles;


// Archivos para controlar resultados:

FILE *salida;
```

```
FILE *frastros;
FILE *fcostos;

int aux_iter=0; // para debuggear
float costo_ant; // se declara global para debuggear
float costo_rh, costo_cap, costo_desp;     //son para debuggear
float costo_oreq, costo_comp, costo_espac,costo_adyac;//son para debuggear



//******** Fin. Determina si el algorimo debe detenerse ********

int fin(int cant_iter)
{
    if (cant_iter > max_iter)
        return(TRUE);

    return(FALSE);

}

// ********** insertar clases de un grupo en la matriz mat_gru *******

void insertar_clases_grupo_en_mat_gru(int car, int anio, int cur, int gru,
                                      vector_sh   sol_sh_clase,
                                      matriz2d_gru &mat_gru)
{
    int fin_clases_gru;
    int cla, ind_cla;
    t_string10 id_clase;
   char id_salon[4];
    int sal, hor_ini, hor_fin, dia, h, h_dia;
    t_elem_cs elem_cs(" ", " ");
    t_elem_cs ant_elem_cs(" ", " ");


    fin_clases_gru=FALSE;
    cla=0;
    while(!fin_clases_gru)
   {
        ind_cla = estructura_facu[car].anios_car[anio].
                  cursos_anio[cur].grupos_cur[gru].clases_gru[cla];
        if (ind_cla!=(-1))
        {
            obtener_id_clase(ind_cla, id_clase);
            sal = sol_sh_clase.consultar_salon(ind_cla);
            obtener_id_salon(sal, id_salon);
            elem_cs.insertar(id_clase, id_salon);
            sol_sh_clase.consultar_hors(ind_cla, &hor_ini, &hor_fin);
            dia = obtener_dia(hor_ini);
            for(h=hor_ini; h<=hor_fin; h++)
            {
                h_dia = obtener_hora_del_dia(h);

                mat_gru.consultar(h_dia, dia, ant_elem_cs);
                ant_elem_cs.cons_id_clase(id_clase);


                if(!cmp_str(id_clase,"  ",1))
                {
                    fprintf(stderr,"Error: se intento insertar en
                              mat_gru[%i][%i] y\n", h_dia, dia);
                    fprintf(stderr,"ya estaba ocupado por: %s\n", id_clase);
                    exit(-1);
                }
```

```
                mat_gru.insertar(h_dia, dia, elem_cs);
            }
        }
        else
            fin_clases_gru=TRUE;
        cla++;
    } // while

}


// ********************** mostrar asignacion por grupos **************

void mostrar_asignacion_por_grupos( vector_sh sol_sh_clase,
                                        matriz3d_str gru_caranio )
{
    matriz2d_gru mat_gru(cant_horas_dia, cant_dias_sem);
    int car, anio, cur, gru;

    // variables para gru_caranio:
    int gru_ca; // indice.
    t_string5 id_gru_ca; // identificador de grupo.
    int fin_grupos_ca; // indica fin de los grupos de una carrera-año.

    char id_carrera[4];
    char id_anio[3];
    char id_curso[5];
    char id_grupo[5];


    fprintf(salida,"\n*******************************\n");
    fprintf(salida,"**** asignacion por grupos ****\n");
    fprintf(salida,"*******************************\n");


    // recorro estructura de la facultad y cargo matriz de grupo con
    // las clases de un grupo:

    for(car=0;car<cant_carreras;car++)
    {
        strcpy(id_carrera, estructura_facu[car].id_carrera);
        for(anio=0;anio<cant_anios;anio++)
        {
            strcpy(id_anio, estructura_facu[car].anios_car[anio].id_anio);
            gru_ca = 0;
            fin_grupos_ca = FALSE;
            while(!fin_grupos_ca)
            {
                gru_caranio.consultar(car, anio, gru_ca, id_gru_ca);
                if ( strlen(id_gru_ca)==0 )
                    fin_grupos_ca = TRUE;
                else
                {
                    mat_gru.inicializar();
                    for(cur=0;cur<cant_cursos;cur++)
                        for(gru=0;gru<cant_grupos;gru++)
                        {
                            strcpy(id_grupo, estructura_facu[car].
                                anios_car[anio].cursos_anio[cur].
                                grupos_cur[gru].id_grupo);
                            if (cmp_str(id_grupo, " ", 1))
                                break;
                            if(cmp_str(id_grupo, id_gru_ca, 4))
                                // es uno de los grupos correspondientes
```

```
                            // al grupo que estoy mostrando ahora
                            insertar_clases_grupo_en_mat_gru(car, anio, cur,
                                gru, sol_sh_clase, mat_gru);
                        } // for grupos
                  } // else fin_grupos_ca

                  // mostrar la planilla de clases de un grupo:
                  if(!fin_grupos_ca)
                {
                     fprintf(salida,"\n****** car: %s año: %s gru: %s\n",
                        id_carrera, id_anio, id_gru_ca);
                     mat_gru.mostrar();
                }

                gru_ca++;
              } // while grupos_caranio
            } // for anios
        } // for carreras

}

// *************** Mostrar resultados ****************

void mostrar_resultados(int encontro_sol, matriz2d solucion,
                            matriz3d_real mat_rastros, int iter_min_costo,
                            vector_sh sol_sh_cla, matriz3d_str gru_caranio)
{
    if(encontro_sol)
    {
        fprintf(salida,"\n\n*****************************\n");
        fprintf(salida,"\n  S O L U C I O N   F I N A L \n");
        fprintf(salida,"\n***************************** \n\n");
        fprintf(salida,"\n Matriz de Asignación: \n\n");
        solucion.mostrar();
        fprintf(salida,"\n\n\n C O S T O S : \n\n");
        fprintf(salida,"El costo de la mejor solución es: %.2f \n",costo_ant);
        fprintf(salida,"costo_rh es : %.4f \n", costo_rh);
        fprintf(salida,"costo_cap es : %.4f \n",costo_cap);
        fprintf(salida,"costo_desp es : %.4f \n", costo_desp);
        fprintf(salida,"costo_oreq es : %.4f \n", costo_oreq);
        fprintf(salida,"costo_comp es : %.4f \n",costo_comp);
        fprintf(salida,"costo_espac es : %.4f \n",costo_espac);
        fprintf(salida,"costo_ady es : %.4f \n", costo_adyac);

        fprintf(salida,"Iteracion de la mejor solución : %i
\n\n",iter_min_costo);

        mostrar_asignacion_por_grupos(sol_sh_cla, gru_caranio);
    }
    else
    {
        fprintf(salida,"\n*****************************\n");
        fprintf(salida,"*** no se encontro solucion ***\n");
        fprintf(salida,"*****************************\n");

        printf("\n*****************************\n");
        printf("*** no se encontro solucion ***\n");
        printf("*****************************\n");

    }


    fprintf(frastros,"mat. rastros definitiva\n");
      mat_rastros.mostrar();
}
```

```
//*********************************************************************
//**********************   M  A  I  N   ******************************
//*********************************************************************


main( int argc, char *argv[] )
{
    // han declarado como globales los datos de la
    // estrucura de la facultad.

    int m;

    if (argc !=2)
    {
     fprintf(stderr,"\nSintaxis de invocacion:\n");
     fprintf(stderr,"ejecutable.exe nom_archivo_índice.txt\n");
     exit(-1);
    }

    salida = fopen("salida.txt","w");
   frastros = fopen("frastros.txt","w");
    fcostos = fopen("fcostos.txt","w");


    carga_de_dimensiones(argv[1]);

    matriz3d_real mat_rastros(cant_salones,cant_hors,cant_clases);

    // matriz con los grupos pertenecientes a una carrera y año.
    // se utiliza para desplegar la asignacion por grupos:
    matriz3d_str gru_caranio(cant_carreras, cant_anios,
                             cant_cursos * cant_grupos);

    carga_de_datos(argv[1], gru_caranio);


    fprintf(salida, "\nen main\n");

    fprintf(salida, "cant_carr %i\n",cant_carreras);
    fprintf(salida, "cant_anios %i\n",cant_anios);
    fprintf(salida, "cant_cursos %i\n",cant_cursos);
    fprintf(salida, "cant_grupos %i\n",cant_grupos);
    fprintf(salida, "cant_clases_gru %i\n",cant_clases_gru);
    fprintf(salida, "cant_clases %i\n",cant_clases);
    fprintf(salida, "cant_salones %i\n",cant_salones);




    // ********* Algoritmo principal: **********



    matriz3d_real mat_rastros_aux(cant_salones, cant_hors, cant_clases);
    matriz2d      solucion(cant_salones, cant_hors);
    matriz2d        mej_sol_iter(cant_salones, cant_hors); // guarda la mejor
                                  // solucion obtenida por una hormiga
dentro
                                  // de una iteracion.


    // para debuggear: solucion_trunc:
    matriz2d      solucion_trunc(cant_salones, cant_hors);
```

```
   vector_sh     sol_sh_cla(cant_clases);

    hormiga coquita(cant_salones,cant_hors,cant_clases);

    float costo;

    //float costo_ant; // se usa global para debuggear

    time_t hora;
    int cant_iter;
    int movio; // indica si la hormiga pudo asignar una clase
                // de lo contrario paso a la siguiente hormiga.

    int encontro_sol; // indica si el algoritmo encontro al menos
                          // una solucion.
    int encontro_sol_iter; // indica si el algoritmo encontro al
                               // menos una solucion en la iteracion actual.

    int i, j, k ;
    t_string10 elem, cla;
    int iter_min_costo;
    float min_costo_iter;   // lleva el costo de la mejor hormiga de
                                // la iteracion actual


    costo_ant=RAND_MAX;


    coquita.resetear();


    srand((unsigned) time(&hora));

    cant_iter=0;

    mat_rastros.inicializar(min_rastro);

   encontro_sol = FALSE;


    do
    {
        min_costo_iter=RAND_MAX;
        mat_rastros_aux.inicializar(0);
        encontro_sol_iter = FALSE;

        for (i=1;i<=cant_horm;i++)  // para cada hormiga
        {
            do
            {
              movio = coquita.mover(mat_rastros);
              aux_iter++;  // para debuggear.
              //fprintf(salida,"coquita.llena lt: %i\n",coquita.llena_lt());
            }
           while ((!coquita.llena_lt())&&(movio));

            if (movio)
            {
            encontro_sol = TRUE;
                encontro_sol_iter = TRUE;

                costo = coquita.calcular_costo_asignacion();
                if ( costo < min_costo_iter ) // guardo el costo de la mejor
hormiga
             {
                    min_costo_iter=costo;
```

```
                    coquita.copiar_asig(mej_sol_iter);
                    //fprintf(salida,"mej_sol_iter\n");
                    //mej_sol_iter.mostrar();
                    //fprintf(salida,"min_costo_iter %.2f\n", min_costo_iter);
             }

               if (fmod(aux_iter, 10)==0)
               {

               fprintf(fcostos,"iteracion: %i  ",cant_iter);
                   fprintf(fcostos,"solucion: %.2f\n", costo);
               }

               if (costo < costo_ant)
               {
                   coquita.copiar_asig(solucion);
                   coquita.copiar_sh_clase(sol_sh_cla);
                   costo_ant = costo;
                iter_min_costo = cant_iter;
               }


             }
             //else  // para debuggear
            //{
             //  coquita.copiar_asig(solucion_trunc);
              // fprintf(salida,"**** solucion truncada:\n");
                //   solucion_trunc.mostrar();
             //}

             coquita.resetear();  // lista tabu, lista sh, asignacion.

          } // para cada hormiga


          // agregar rastros dejados por la MEJOR hormiga en esta iteracion
          // a la matriz de rastros y actualizarla segun el coeficiente
          // de evaporacion:

         if (encontro_sol_iter) // si en esta iteracion se encontro alguna
solucion
             mat_rastros.actualizar_iter(mej_sol_iter, min_costo_iter);


          if (fmod(cant_iter, 100)==0)
         {
          fprintf(frastros,"mat. rastros definitiva en iter: %i\n",cant_iter);
             mat_rastros.mostrar();
          }


          cant_iter++;
          if (fmod(cant_iter, 10)==0)
             printf("Iteracion : %i\n",cant_iter);


      }
      while (!fin(cant_iter));

      mostrar_resultados(encontro_sol, solucion, mat_rastros,
                           iter_min_costo, sol_sh_cla, gru_caranio);

      fclose(frastros);
      fclose(salida);
      fclose(fcostos);
```

```
    printf("Terminé. Andá a leer el archivo \n");
    return 0;
}
```

Cambios en los fuentes del algoritmo para implementar mejoras a las funciones de visibilidad y rastros

Hormiga.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <alloc.h>
#include "auxiliar.hpp"
#include "matrices.hpp"
#include "hormiga.hpp"
#include "estrfacu.hpp"

#define RAND rand()/RAND_MAX
#define VACIO -1
#define NO_ENCONTRE -1
#define LIBRE -1



extern float costo_max;
extern float costo_min;
extern float gama;
extern float costo_vis;
extern float max_cvis, c1;



extern float max_costo;
extern float mult_rastro;
extern float min_visib;
extern float max_visib;
extern float min_prob;


extern t_carreras_fac estructura_facu;
//vector  con todas las clases de facultad:
extern t_clases_fac clases_fac;
extern t_vect_turnos v_turnos;
extern t_salones salones_fac;
//extern t_perfiles perfiles;

extern int aux_iter;
extern FILE *salida;
extern FILE *fcostos;
extern FILE *frastros;

extern int cant_hors;
extern int cant_salones;
extern int cant_clases;
extern int cant_grupos;
extern int cant_cursos;
extern int cant_anios;
extern int cant_carreras;
extern int cant_horas_dia;
extern int cant_clases_gru;
extern int cant_dias_sem;

extern float exp_rastro;
extern float exp_visib;
```

```
extern float peso_ran_hor;
extern float peso_capac_salon;
extern float peso_otros_req;
extern float peso_desper_salon;
extern float peso_espaciamiento;
extern float peso_adyacencia;
extern float peso_compactibilidad;


extern float costo_ant ,costo_rh, costo_cap, costo_desp;
extern float costo_oreq, costo_comp, costo_espac,costo_adyac;




int vez=0;
int vez_recta=0;


int obtener_dia(int hor)
{
    div_t x;

    x = div(hor,cant_horas_dia);
   return( x.quot);

}

int obtener_hora_del_dia(int hor)
{
 return( fmod(hor, cant_horas_dia));
}



int obtener_turno(int hor)
{

    int i;
    int h_ini_dia, h_dia;
   int tur_ini, tur_fin;

    h_dia = obtener_hora_del_dia( hor);

    for(i=0;i<v_turnos.tope;i++)
    {
        tur_ini=v_turnos.vect[i].h_ini;
        tur_fin=v_turnos.vect[i].h_fin;
        if((tur_ini <= h_dia)&&(h_dia <= tur_fin))
        return(v_turnos.vect[i].turno);
    }

    return(-1);

}



// *************** hormiga ************************

hormiga::hormiga( int n_salones, int n_horarios, int n_clases)
{
  lista_tabu = vector_tope(n_clases);
  lista_sh = matriz2d_indice(n_salones, n_horarios);
```

```
  asignacion = matriz2d(n_salones, n_horarios);
  sh_clases = vector_sh(n_clases);
  hors_caranio = mat_horscaranio(cant_carreras, cant_anios, cant_hors);

}

void hormiga::resetear()
{
 lista_tabu.resetear();
 lista_sh.resetear();
 asignacion.inicializar();
 sh_clases.inicializar();
 hors_caranio.inicializar();
}



int hormiga::consultar_lt(int d)
{
  int el;
  el=lista_tabu.consultar(d);
  return(el);
}


void hormiga::insertar_lt(int d)
{
  lista_tabu.insertar(d);
}


int hormiga::llena_lt()
{
    return(lista_tabu.llena());
}


int hormiga::consultar_sh(int d1, int d2)
{
  return (lista_sh.consultar(d1, d2));
}


void hormiga::insertar_sh(int d1, int d2)
{
  lista_sh.insertar(d1, d2);
}



int hormiga::consultar_asig(int d1, int d2)
{
    return(asignacion.consultar(d1, d2));
}


void hormiga::insertar_asig(int d1, int d2, int elem)
{
  asignacion.insertar(d1, d2, elem);
}


void hormiga::copiar_asig(matriz2d mat)
{
       int sal, hor;
      int dim1, dim2;
```

```
       asignacion.consultar_dims(&dim1, &dim2);

       for(sal=0;sal<dim1;sal++)
           for(hor=0;hor<dim2;hor++)
               mat.insertar(sal, hor, asignacion.consultar(sal, hor));
}



int hormiga::cons_sh_salon(int cla)
{
 return(sh_clases.consultar_salon(cla));
}


void hormiga::cons_sh_hors(int cla, int *hor_ini, int *hor_fin)
{
 sh_clases.consultar_hors(cla, hor_ini, hor_fin);
}


void hormiga::ins_sh_clase(int cla, int salon, int hor_ini, int hor_fin)
{
 sh_clases.insertar(cla, salon, hor_ini, hor_fin);
}


void hormiga::copiar_sh_clase(vector_sh vec )
{
       int cla, sal, hor_ini, hor_fin;
      int dim1;

       dim1=sh_clases.consultar_dim();

       for(cla=0;cla<dim1;cla++)
      {
           sal=sh_clases.consultar_salon(cla);
         sh_clases.consultar_hors(cla,&hor_ini,&hor_fin);
           vec.insertar(cla, sal, hor_ini, hor_fin);
      }

}



void asignar(vector_tope &lista_tabu, matriz2d_indice lista_sh,
                    mat_horscaranio hors_caranio, vector_sh sh_clases,
                 matriz2d asignacion,
                    int sal, int hor, int cla)
{

    int dur, h, u;
    t_vect_ubicacion v_ubic;
   int fil, col;


    clases_fac[cla].consultar_ubicacion(v_ubic);
    dur=clases_fac[cla].consultar_durac();

    for(h=hor;h<(hor+dur);h++)
    {
     lista_sh.insertar(sal, h);
     asignacion.insertar(sal, h, cla);
     for(u=0;(u<v_ubic.tope);u++) // para cada ubicacion de la clase
     {
```

```
        hors_caranio.insertar(v_ubic.vect[u].ind_car, v_ubic.vect[u].ind_anio,
h,
                                    v_ubic.vect[u].grupo);
     }
    }

    sh_clases.insertar(cla, sal, hor, hor+dur-1);

    lista_tabu.insertar(cla);

    return;
}




// **************************** Asignar en Bloque
*****************************

void asignar_bloque(vector_tope &lista_tabu, matriz2d_indice lista_sh,
                     mat_horscaranio hors_caranio, vector_sh sh_clases,
                     matriz2d asignacion,
                     int hor, int cla)
{
    t_vect_ubicacion v_ubic;
    int *clases_grupo; // alias
   int cla_gru;
   int i_car, i_anio, i_cur, i_gru;
    int hor_dia, h_ini, h_fin, dia, hor_tent, hor_tent_ini, sal_tent;
    int i,j,k;
    rangos prefs;
    int cumple_rest;
    int fil, col, ind;

    int azar;
    vector_tope_entero vect_sals_tent(20);
    vector_tope_entero vect_hors_tent(20);

    clases_fac[cla].consultar_ubicacion(v_ubic);

    for(k=0;(k<v_ubic.tope);k++) // para cada ubicacion de la clase
    {
        i_car  = v_ubic.vect[k].ind_car;
        i_anio = v_ubic.vect[k].ind_anio;
        i_cur  = v_ubic.vect[k].ind_cur;
        i_gru  = v_ubic.vect[k].ind_gru;

        // obtener clases del grupo de la clase que quiero asignar:
        clases_grupo =
estructura_facu[i_car].anios_car[i_anio].cursos_anio[i_cur].grupos_cur[i_gru]
.clases_gru;

        j=0;
        while (clases_grupo[j]!=(-1))   // mientras hayan clases en el grupo
        {
            cla_gru=clases_grupo[j];
            if((cla_gru!=cla)&&(clases_fac[cla_gru].consultar_es_teo()==TRUE))
            // que no sea la clase que estoy
            {                           // controlando

                sh_clases.consultar_hors(cla_gru, &h_ini, &h_fin);
                if (h_fin == 0) // la clase no fue asignada todavía
                {
                    vect_hors_tent.resetear();
                    vect_sals_tent.resetear();
```

```
                   ind=0; //indice para vectores

                   // obetenemos el horario correspondiente al primer dia de la
semana
                   // para la clase original:
                hor_tent_ini = fmod(hor, cant_horas_dia);

                   // obtenemos rangos horarios de preferencia de la clase
                   arrcpy1(prefs, clases_fac[cla_gru].pref_hors);

                   for(dia=0; dia < cant_dias_sem; dia++)
                   {
                       hor_tent = hor_tent_ini + dia*cant_horas_dia;
                       i=0;
                       while(prefs[i].hor_ini != prefs[i].hor_fin)
                   {
                           if ((prefs[i].hor_ini <= hor_tent)
                                 && (prefs[i].hor_fin >= hor_tent ))
                           {
                               for(sal_tent=0; sal_tent < cant_salones;
sal_tent++)
                               {

                                   cumple_rest = cumple_restricciones(lista_tabu,
lista_sh,
                                    hors_caranio, sh_clases, sal_tent, hor_tent,
cla_gru );

                                   if (cumple_rest)
                                   {
                                       vect_hors_tent.insertar(ind,hor_tent);
                                       vect_sals_tent.insertar(ind,sal_tent);
                                       ind++;
                                 }

                                  }
                           }
                       i++;
                   } // while
                   } // for dia

                   if(vect_hors_tent.consultar_tope() > 0)
                {
                       // sorteo una de las asignaciones tentativas:
                       azar=(float)RAND*(vect_hors_tent.consultar_tope() -1 );
                       sal_tent = vect_sals_tent.consultar(azar);
                       hor_tent = vect_hors_tent.consultar(azar);

                       lista_sh.consultar_indices(&fil,&col);

                       asignar(lista_tabu,lista_sh, hors_caranio, sh_clases,
                                 asignacion, sal_tent, hor_tent, cla_gru);
                   }

               } // if no fue asignada
           } // if
          j++;
      } // while clases grupo
   } // for ubicaciones

   vect_hors_tent.destr_vector();
   vect_sals_tent.destr_vector();

}
```

```
//********************** dejar rastro ***************************

void hormiga::dejar_rastro(float costo, matriz3d_real mat_rastro)
{
    float rastro;
    int sal, hor, cla;
    int dim1, dim2;
    float f0, f1, coef_m, coef_n;


    if(costo>max_costo)
    {
        fprintf(salida,"costo > MAX_COSTO\n");
        exit(-1);
    }



    f0=15;
    f1=0.05;

    coef_m=(f0-f1)/((1/pow(costo_min,2))-(1/pow(costo_max,2)));

   coef_n=f1- (coef_m/pow(costo_max,2));


   rastro=(coef_m/pow(costo,2))+coef_n;

    asignacion.consultar_dims(&dim1, &dim2);

    for(sal=0;sal<dim1;sal++)
        for(hor=0;hor<dim2;hor++)
        {
            cla = asignacion.consultar(sal, hor);
          if (cla != -1)
               mat_rastro.acumular(sal, hor, cla, rastro);
      }

}



// ******************** mover hormiga ***********************



int obtener_siguiente_sh(matriz2d_indice lista_sh, int *p_sal, int *p_hor)
{
   int res;
    int ya_asignado=TRUE;
   int fin_matriz=FALSE;
    while ( (ya_asignado==TRUE) && (fin_matriz==FALSE) )
    {
      if (lista_sh.llena()==TRUE)
         fin_matriz=TRUE;
      else
      {
        lista_sh.consultar_indices(p_sal, p_hor);

        if (lista_sh.asignado(*p_sal,*p_hor)==TRUE)
        {
            lista_sh.act_indices();
```

```
         }
         else
              ya_asignado=FALSE;
      }
    }
    return(fin_matriz);

}



//*********************** Restricciones *********************************


//****** Controlar lista tabu     *****************


int controlar_lista_tabu(vector_tope lista_tabu, int cla)
{

    if ((lista_tabu.consultar(cla))==1)
           return(FALSE);

   return(TRUE);

}



//****** Controlar duracion de la clase*******

int controlar_duracion_clase(matriz2d_indice lista_sh,
                                    int sal, int hor, int cla)
{
   int duracion, ind_hor;

    duracion=clases_fac[cla].consultar_durac();
    ind_hor=hor;
    while ((!lista_sh.consultar(sal,ind_hor))
            &&((ind_hor-hor)<duracion)&&(ind_hor<cant_hors))
        ind_hor++;
    if ( ind_hor-hor!=duracion )
    {
        return(FALSE);
    }

   return(TRUE);
}



//*********** Una clase debe comenzar y terminar dentro del mismo dia *****

int controlar_clase_dentro_dia(int hor, int cla)
{
    int dia_ini, dia_fin;
   int duracion;

    duracion=clases_fac[cla].consultar_durac();
    dia_ini=obtener_dia(hor);
    dia_fin=obtener_dia(hor+duracion-1);
    if (dia_ini!=dia_fin)
    {
        return(FALSE);
    }
```

```
   return(TRUE);
}



//******** Las clases de un mismo perfil no deben superponerse

/*
int controlar_perfiles(t_perfiles perfiles,
                           int hor, int cla)
{

    int h_ini, h_fin;
    int superpone;
    int p,c,c2;
    int dur;


    dur=clases_fac[cla].consultar_durac();
    for(p=0;p<perfiles.tope;p++) // recorro perfiles de la facultad.
    {
        vector_tope_entero clases_perf;
        clases_perf=perfiles.vect[p];
        c=0;
        while ((
clases_perf.consultar(c)!=cla)&&(c<clases_perf.consultar_tope()))
             c++;
        if( c<clases_perf.consultar_tope())  // la clase esta en el perfil
        {
            c2=0;
            superpone=FALSE;
        while ((c2<clases_perf.consultar_tope()) && (!superpone))
            {
                sh_clases.consultar_hors(clases_perf.consultar(c2),
                            &h_ini, &h_fin);
                if ((h_ini<=hor && h_fin>=hor)||
                    (hor<=h_ini && (hor+dur-1)>=h_ini))
                    superpone=TRUE;
                else
                    c2++;
            }
        }
        if (superpone)
             return(FALSE);

   } // end for

    return(TRUE);
 }
 */


//*********** Controlar superposicion **************

int controlar_superposicion_grupos(mat_horscaranio hors_caranio,
                                          int hor, int cla)
{
    //**********Los cursos de una misma carrera y año no deben superponerse
********

    // Pero sí pueden admitirse superposiciones entre clases de
    // un mismo curso y distintos grupos (por ej. 2 grupos de
    // practico de An.II que tienen clase a la misma hora en
    // distintos salones.
```

```
    // La superposicion se controla a nivel de grupos.
    // Clases que pertenecen a un mismo grupo no deben superponerse,
    // aún cuando pertenezcan a distintos cursos.


    t_vect_ubicacion v_ubic;
    int i_car, i_anio, i_cur, i_gru;
    t_string5 id_gru;
    int pert_ids_grupo;
    int duracion, ind_hor;
   int i;


    duracion = clases_fac[cla].consultar_durac();
   clases_fac[cla].consultar_ubicacion(v_ubic);
    for(i=0;i<v_ubic.tope;i++) // recorro ubicaciones de la clase.
    {
        i_car  = v_ubic.vect[i].ind_car;
        i_anio = v_ubic.vect[i].ind_anio;
        strcpy(id_gru, v_ubic.vect[i].grupo);

        for(ind_hor=hor; ind_hor<(hor+duracion); ind_hor++) // recorro
horarios
        {                                                              // a los
que quiero asignar
                                                                       // la
clase
           if (ind_hor==cant_hors) // fin de la matriz de asignacion
           return(FALSE);
           else
        {
               pert_ids_grupo=hors_caranio.pertenece(i_car, i_anio, ind_hor,
id_gru);
               if (pert_ids_grupo) // ya se ha asignado una clase con el mismo
grupo
           {
                   return(FALSE);
               }
           } // end else
        } // end for
    } // end for

   return(TRUE);
}



//************* Controlar clases en su turno *********************

int controlar_clase_en_su_turno(int hor, int cla)
{
    //Controlar que las clases de un grupo se asignen en el turno
    //indicado por este.

    int turno_grupo, turno_tentativo_fin, turno_tentativo_com;
    t_vect_ubicacion v_ubic;
    int duracion;
    int i_car, i_anio, i_cur, i_gru;
   int i;


    duracion=clases_fac[cla].consultar_durac();

    clases_fac[cla].consultar_ubicacion(v_ubic);
    for(i=0;i<v_ubic.tope;i++) // recorro ubicaciones de la clase.
```

```
    {

        i_car  = v_ubic.vect[i].ind_car;
        i_anio = v_ubic.vect[i].ind_anio;
        i_cur  = v_ubic.vect[i].ind_cur;
        i_gru  = v_ubic.vect[i].ind_gru;

        turno_grupo =
estructura_facu[i_car].anios_car[i_anio].cursos_anio[i_cur].grupos_cur[i_gru]
.turno;
       turno_tentativo_com = obtener_turno(hor); //la clase debe comenzar
dentro del turno
       turno_tentativo_fin = obtener_turno(hor+duracion-1); //la clase debe
terminar dentro del turno

       if
((turno_grupo!=turno_tentativo_com)||(turno_grupo!=turno_tentativo_fin))
       {
           return(FALSE);
    }

    } // for

    return(TRUE);

}



//************* Controlar separacion de las clases de un grupo
*********************

int controlar_separacion_clases_grupo(vector_sh sh_clases, int hor, int cla)
{

    //****   No pueden asignarse 2 clases de un mismo grupo el mismo dia
    // a no ser que una clase sea de teorico y la otra de práctico.

    int *clases_grupo; //es un alias ojo!!
    t_vect_ubicacion v_ubic;
   int dia_asignado, dia_tentativo;
    int cla_gru;
    int es_teo_cla, es_teo_cla_gru;
    int i_car, i_anio, i_cur, i_gru;
    int i, j;
    int h_ini, h_fin;


    //fprintf(salida,"*********controlando clase %i\n",cla);

    es_teo_cla=clases_fac[cla].consultar_es_teo();
    clases_fac[cla].consultar_ubicacion(v_ubic);
    for(i=0;i<v_ubic.tope;i++) // recorro ubicaciones de la clase.
    {

        i_car = v_ubic.vect[i].ind_car;
        i_anio = v_ubic.vect[i].ind_anio;
        i_cur = v_ubic.vect[i].ind_cur;
        i_gru = v_ubic.vect[i].ind_gru;

        clases_grupo =
estructura_facu[i_car].anios_car[i_anio].cursos_anio[i_cur].grupos_cur[i_gru]
.clases_gru;

        j=0;
        while( clases_grupo[j]!=(-1)) // mientras hayan clases en el grupo
```

```
        {
            if(clases_grupo[j]!=cla) // que no sea la clase que estoy
controlando
            {
            cla_gru=clases_grupo[j];
                es_teo_cla_gru=clases_fac[cla_gru].consultar_es_teo();

                if (es_teo_cla_gru==es_teo_cla)
            {
                    sh_clases.consultar_hors(clases_grupo[j], &h_ini, &h_fin);
                    if (h_ini!=h_fin) // si la clase ya fue asignada
                    {
                        dia_asignado=obtener_dia(h_ini);
                        dia_tentativo=obtener_dia(hor);
                        if(dia_asignado==dia_tentativo)
                    {
                            return(FALSE);
                    }
                }
                }
            }
            j++;
        } // while
    } // for ubicaciones

    return(TRUE);

}



//********************* cumple restricciones ************************

int cumple_restricciones(vector_tope lista_tabu, matriz2d_indice lista_sh,
                             mat_horscaranio hors_caranio, vector_sh
sh_clases,
                             int sal, int hor, int cla)
{
    int cumple;


    cumple = controlar_lista_tabu(lista_tabu, cla);
    if(!cumple)
        return(FALSE);

   cumple = controlar_duracion_clase(lista_sh, sal, hor, cla);
    if(!cumple)
        return(FALSE);

    cumple = controlar_clase_dentro_dia(hor, cla);
    if(!cumple)
        return(FALSE);

    //cumple = controlar_perfiles(perfiles, hor, cla);
    //if(!cumple)
        //return(FALSE);

    cumple = controlar_superposicion_grupos(hors_caranio, hor, cla);
    if(!cumple)
        return(FALSE);


    cumple = controlar_clase_en_su_turno(hor, cla);
    if(!cumple)
        return(FALSE);
```

```
    cumple = controlar_separacion_clases_grupo(sh_clases, hor, cla);
   if(!cumple)
       return(FALSE);

   //****   Recursos móviles ( retroproyectores )


    return(TRUE);

}




//****************Visibilidades*************************************



 //****** Controlar que el salon satisfaga otros requerimientos
 //************ version para calculo de visibilidad **************



float controlar_otros_req_vis(int sal, int cla)
{
 float costo;
 int i;
 vectint otras_car, otras_pref;

 costo=0;

 salones_fac[sal].consultar_otras_car(otras_car);
 clases_fac[cla].consultar_otras_pref(otras_pref);
 for(i=0;i<10;i++)
 {
    if ((otras_pref[i]==1) && (otras_car[i]==0))
              costo++;

 }

 return(costo);
}



//************ Controlar desperdicio de la capacidad del salon ******
//************ version para calculo de visibilidad **************



float controlar_desper_salon_vis(int sal, int cla )
{
    float costo;
   float sobran;
    float pct;

    costo=0;


    sobran=salones_fac[sal].capacidad-clases_fac[cla].cant_estud;
    if (sobran >0)
   {
         pct=(sobran/salones_fac[sal].capacidad)*100;
         if (pct >30)
```

```
            costo++;

    }
    return(costo);
}



//************ Controlar capacidad del salon suficiente ******
//************ para la clase.                           ******
//************ version para calculo de visibilidad **************


float controlar_capac_salon_vis(int sal, int cla)
{
    float costo;

    costo=0;
    if (clases_fac[cla].cant_estud > salones_fac[sal].capacidad)
                  costo=1;
    return(costo);
}



//************ Controlar rangos horarios de preferencia *********
//************ version para calculo de visibilidad **************

float controlar_rangos_horarios_vis(int hor_ini, int cla)
{
    int dist, dist_min, total_dist=0;
    int duracion, hor_fin;
    rangos prefs;
   int i;

     // consultar duracion de cla y obtener hor_fin:

     duracion = clases_fac[cla].consultar_durac();
     hor_fin = hor_ini + duracion;

     // obtenemos rangos horarios de preferencia de la clase

     arrcpy1(prefs,clases_fac[cla].pref_hors);
     dist_min=cant_hors;
     if ( prefs[0].hor_ini==prefs[0].hor_fin )
        dist_min=0;
     else
     {
      i=0;

      while((i<10) && (!( prefs[i].hor_ini==prefs[i].hor_fin)))
      {
       if ( hor_fin > prefs[i].hor_fin )
            dist= hor_fin-prefs[i].hor_fin;
       else
        if ( hor_ini < prefs[i].hor_ini )
           dist= prefs[i].hor_ini- hor_ini;
        else
           dist=0;
       if ( dist<dist_min )
          dist_min=dist;
       i++;
      }
     }
     total_dist+=dist_min;
```

```
    return(total_dist);
}



//*************** Calcular visibilidad **********************

float calcular_visibilidad(int sal, int hor, int cla)
{
    float costo,visib;

    costo = peso_ran_hor * controlar_rangos_horarios_vis(hor, cla);
    costo+= peso_capac_salon * controlar_capac_salon_vis(sal, cla);
    costo+= peso_otros_req * controlar_otros_req_vis( sal, cla);
    costo+= peso_desper_salon * controlar_desper_salon_vis( sal, cla );

    if ( costo> max_cvis )
        max_cvis=costo;


    if (costo > max_costo)
    {
        fprintf(stderr,"Costo muy alto en calculo de visibilidad\n");
        exit(-1);
    }


    float c0,x1,f0,f1,f2,dcero,duno,ddos,ecero,euno,edos;
    float coef_b, coef_a, coef_c;



    c0=5 ;
    c1=100;
    x1=19*peso_ran_hor;

    f0=1;
    f1=0.05;
    f2=0.19;


    if (( c1>x1 ) && ( c0<x1))
    {
      dcero=(1/(c0*c0))-(1/(c1*c1));
      duno=(1/c0)-(1/c1);
      ddos=f0-f1;
      ecero=(1/(c1*c1))-(1/(x1*x1));
      euno=(1/c1)-(1/x1);
      edos=f1-f2;

      coef_b=(ddos+edos)/(duno-(dcero*euno/ecero));

      coef_a=(ddos-(coef_b*duno))/dcero;


      coef_c=f0- (coef_a/(c0*c0))-(coef_b/c0);

    }

    if ( costo<=c0)
        visib=f0;
    else
      if ( costo==c1)
          visib=f1;
      else
```

```
          visib= (coef_a/pow(costo,2))+(coef_b/costo)+coef_c;


    if (fmod(aux_iter,20)==0 )
    {
        c1=max_cvis;
        max_cvis=0;
    }




   return(visib);
}



// ************************ Calcular probabilidad ************************


float calcular_prob(int sal, int hor,int cla, matriz3d_real mat_rastros)
{
 //La probabilidad de una clase depende del rastro y la visibilidad.

 float rastro, visib, prob;

 rastro=mat_rastros.consultar(sal, hor, cla);

 visib =calcular_visibilidad(sal, hor, cla);

 // si la visibilidad de una clase es muy baja, se pone en cero
 // para que la clase no sea considerada. De este modo, si no
 // existe ninguna clase atractiva para el (s,h), la hormiga no
 // encontrará clases para asignar, y deberá avanzar al siguiente
 // (s,h).
 // Si todas las visibilidades fueran bajas, pero no cero, de todos
 // modos se elegirá alguna clase, no avanzando el (s,h).

 if (visib < min_visib)
    visib = 0;

 prob = pow(rastro,exp_rastro) * pow(visib,exp_visib);

 return(prob);



}


int asignar_clase()
{
 int max_sorteo=100;
 float azar;

 azar=(float)RAND*max_sorteo;
 if (azar<=1)
    return(TRUE);
 else
    return(FALSE);
}
```

```
//************************ elegir clase ******************************

void elegir_clase(vector_tope lista_tabu, matriz2d_indice lista_sh,
                      mat_horscaranio hors_caranio, vector_sh sh_clases,
                      matriz3d_real mat_rastros,
                      int sal, int hor, int *p_cla)
{
    // elige la clase a asignar a la pareja (sal, hor) en base a
    // la visibilidad y los rastros.
    // la visibilidad depende de las restricciones y preferencias
    // locales.

  int j;
  float suma;
  float *recta;
  float *recta_sacum;
  float azar;
  float prob;
  float max_prob;

  int elijo;


     recta=(float*)calloc(cant_clases,sizeof(float));

     if (recta==NULL)
     {
          fprintf(salida,"No se pudo asignar memoria para recta \n");
          exit(-1);
     }

     recta_sacum=(float*)calloc(cant_clases,sizeof(float));

     if (recta_sacum==NULL)
     {
          fprintf(salida,"No se pudo asignar memoria para recta_sacum \n");
          exit(-1);
     }


     suma=0;

    max_prob=0;
     for(j=0;j<cant_clases;j++)
     {
       if ( cumple_restricciones(lista_tabu, lista_sh, hors_caranio,
sh_clases,
                                       sal, hor, j))
       {
           prob=calcular_prob( sal, hor,j, mat_rastros); //dep de rastro y
visibilidades
           if ( prob>max_prob)
             max_prob=prob;

          recta[j]=prob;
       }
       else
       {
           recta[j]=0;
       }
     }
```

```
     // Se construye la recta de probabilidades para sortear una clase.
     // Se normaliza la misma para que contenga valores entre 0 y 1.


     if (max_prob==0)
     {
         (*p_cla)=NO_ENCONTRE;
         free(recta_sacum);
        free(recta);
         return;
    }
     else
     {
      suma=0;
       for(j=0;j<cant_clases;j++)
       {
         recta[j]=recta[j]/max_prob;
         recta_sacum[j]=recta[j];
         suma+=recta[j];
       }
    }

     // se acumulan los valores de la recta:

    for(j=1;j<cant_clases;j++)
    {
      recta[j]=recta[j]+recta[j-1];
     }

    // genero un numero al azar entre 0 y suma

     azar=(float)RAND*suma;

    *p_cla=0;
    while((azar>recta[*p_cla])||(recta[*p_cla]==0))
        (*p_cla)++;

     // si el valor de la recta (sin acumular) para la clase sorteada
     // es menor que min_prob, la clase se descarta:

     if (recta_sacum[*p_cla]<min_prob)
         (*p_cla=NO_ENCONTRE);



    free(recta_sacum);
     free(recta);

}



void obtener_id_clase(int cla, t_string10 id_clase)
{
    strcpy(id_clase, clases_fac[cla].id_clase);
}




// ********************** mover ***********************************

int hormiga::mover(matriz3d_real mat_rastros)
{
    // devuelve 1 si la hormiga pudo moverse, y
```

```
    // devuelve 0 si no pudo mover.

    int sal, hor, cla ,fin_matriz;
    int dur, i, j;
    t_string10 id_clase;
    int obtuve_clase;
    t_vect_ubicacion v_ubic;
    int res;
    int fil,col;

    fin_matriz=FALSE;
    obtuve_clase=FALSE;
    while ((!obtuve_clase)&&(!fin_matriz))
    {
       fin_matriz=obtener_siguiente_sh(lista_sh, &sal, &hor);
       if (fin_matriz==FALSE)
      {
         elegir_clase(lista_tabu, lista_sh, hors_caranio, sh_clases,
                        mat_rastros, sal, hor, &cla);
        if (cla==NO_ENCONTRE)
        {
           if (lista_sh.llena()==TRUE)
               fin_matriz = TRUE;
           else
           lista_sh.act_indices();
        }
        else
        {
           obtuve_clase=TRUE;
        }

      }
    }


    if (fin_matriz)
    {
     return(0);
    }

    clases_fac[cla].consultar_ubicacion(v_ubic);
    dur=clases_fac[cla].consultar_durac();

    for(i=hor;i<(hor+dur);i++)
    {
     lista_sh.insertar(sal, i);
     asignacion.insertar(sal, i, cla);
     for(j=0;(j<v_ubic.tope);j++) // para cada ubicacion de la clase
     {
        hors_caranio.insertar(v_ubic.vect[j].ind_car, v_ubic.vect[j].ind_anio,
i,
                                    v_ubic.vect[j].grupo);
     }
    }

    sh_clases.insertar(cla, sal, hor, hor+dur-1);

    lista_tabu.insertar(cla);
    if ( clases_fac[cla].consultar_es_teo()==TRUE)
    {
        asignar_bloque(lista_tabu, lista_sh, hors_caranio,
                       sh_clases, asignacion, hor, cla);
        lista_sh.ajustar_indices( sal, hor+dur-1); // ir al siguiente sh
luego de
                                                // asignar la ultima 1/2 hora de la
primera clase
```

```
    }
    return(1);
}


//***************** Puedo controlar *********************
// Indica si corresponde
// controlar espaciamiento en dias entre clases de un mismo grupo.

int puedo_controlar(int cant_cla, int dia_min, int dia_max )
{
  int dist;
  int puedo;

  dist=dia_max-dia_min;


  puedo=1;

  if (cant_cla==1)
  {
     puedo=0;
     return(puedo);
  }

  if ((cant_cla==2) &&(dist<2))
  {
       puedo=0;
       return(puedo);
  }

  if ((cant_cla==3) &&(dist<3))
  {
       puedo=0;
       return(puedo);
  }

  if ((cant_cla==4) &&(dist<4))
  {
       puedo=0;
       return(puedo);
  }

  if (cant_cla==5)
  {
       puedo=0;
       return(puedo);
  }

  return( puedo);
}



//***************** evaluo espacimiento ******************

float evaluo_espaciamiento(int cant_cla, int dia_min, int dia_max )
{
  int dist;
  float costo;

  dist=dia_max-dia_min;

  costo=0;

  if (cant_cla==1)
```

```
  {
     costo=0;
     return(costo);
  }
  if ((cant_cla==2) &&(dist<2))
  {
       costo=1;
       return(costo);
  }


  if ((cant_cla==3) &&(dist<3))
  {
       costo=1;
       return(costo);
  }

  if ((cant_cla==4) &&(dist<4))
  {
       costo=1;
       return(costo);
  }

  if (cant_cla==5)
  {
       costo=0;
       return(costo);
  }


  return( costo);
}



//*********  Controlar Espaciamiento *************


float controlar_espaciamiento(vector_sh sh_clase, int car, int anio,
                                    int cur, int gru)
{
 float costo, eval;
 int cla;
 int fin_clases_gru, dia_min, dia_max, dia, cant_cla_gru;
 int h_ini, h_fin, ind_cla, dist;
 rangos prefs;
 int i, min_hini, max_hfin;

    cla=0;
    fin_clases_gru=FALSE;
    dia_min=6;
    dia_max=0;
    cant_cla_gru=0;
   costo = 0;

    // controlo si corresponde exigir espac de acuerdo a los rangos horarios
    min_hini=cant_hors;
    max_hfin=0;

   while(!fin_clases_gru)
   {
       ind_cla=
estructura_facu[car].anios_car[anio].cursos_anio[cur].grupos_cur[gru].clases_
gru[cla];
       if (ind_cla!=(-1))
```

```
      {
           cant_cla_gru++;
           arrcpy1(prefs,clases_fac[ind_cla].pref_hors);
           i=0;
           while ((i<10)&&(prefs[i].hor_ini!=prefs[i].hor_fin))
           {
             if (prefs[i].hor_ini<min_hini)
                min_hini=prefs[i].hor_ini;
             if (prefs[i].hor_fin>max_hfin)
                 max_hfin=prefs[i].hor_fin;
             i++;
         }

      }
       else
          fin_clases_gru=TRUE;
       cla++;

   } // while


   if (cant_cla_gru !=0)
   {
       dia_min=obtener_dia(min_hini);
       dia_max=obtener_dia(max_hfin);

       if (puedo_controlar(cant_cla_gru,dia_min,dia_max))
       {
           // Obtiene cantidad de clases y primer y ultimo dia de clases en
          // la semana para un grupo:

           cla=0;
          fin_clases_gru=FALSE;
          dia_min=6;
          dia_max=0;
          cant_cla_gru=0;

          while(!fin_clases_gru)
         {
              ind_cla=
estructura_facu[car].anios_car[anio].cursos_anio[cur].grupos_cur[gru].clases_
gru[cla];
              if (ind_cla!=(-1))
              {
                  cant_cla_gru++;
                  sh_clase.consultar_hors(ind_cla,&h_ini,&h_fin);

                  dia=obtener_dia(h_ini);
                  if (dia<dia_min)
                      dia_min=dia;
                  else
                    if (dia > dia_max)
                      dia_max=dia;
           }
               else
                  fin_clases_gru=TRUE;
              cla++;
          } // while
           eval=0;
           eval=evaluo_espaciamiento(cant_cla_gru,dia_min,dia_max);
           costo=costo+eval;
       } // if puedo controlar
   }

   return( costo);
}
```

```
//****** Controlar  si el salon satisface otros requerimientos ******


float controlar_otros_req(vector_sh sh_clases)
{
 float costo;
 int sal, cla;
 int i;
 vectint otras_car, otras_pref;

 costo=0;

 for(cla=0;cla<cant_clases;cla++)
 {
    sal=sh_clases.consultar_salon(cla);
    salones_fac[sal].consultar_otras_car(otras_car);
    clases_fac[cla].consultar_otras_pref(otras_pref);
    for(i=0;i<10;i++)
    {
       if ((otras_pref[i]==1) && (otras_car[i]==0))
            costo++;
    }
 }

 return(costo);
}


//************ Controlar desperdicio de la capacidad del salon ******


float controlar_desper_salon(vector_sh sh_clases)
{
   float costo;
   int sal, cla;
   float sobran;
   float pct;

   costo=0;

   for(cla=0; cla<cant_clases; cla++)
   {
          sal = sh_clases.consultar_salon(cla);
          sobran=salones_fac[sal].capacidad-clases_fac[cla].cant_estud;
          if (sobran >0 )
        {
           pct=(sobran/salones_fac[sal].capacidad)*100;

           if (pct >30)
              costo++;
          }

   }

   return(costo);
}


//********* Controlar que el salon tenga capacidad suficiente *********


float controlar_capac_salon(vector_sh sh_clases)
```

```
{
    float costo;
    int sal, cla;

    costo=0;

    for(cla=0; cla<cant_clases; cla++)
    {
           sal = sh_clases.consultar_salon(cla);
           if (clases_fac[cla].cant_estud > salones_fac[sal].capacidad)
               costo++;
    }

    return(costo);
}




//************ Controlar rangos horarios de preferencia *****************


float controlar_rangos_horarios(vector_sh sh_clases)
{
    int dist , dist_min, total_dist=0;
    int hor_ini, hor_fin;
    rangos prefs;
   int cla, i;


    for(cla=0;cla<cant_clases;cla++)
    {
     //*** obtenemos horario que se le asigno a la clase

     sh_clases.consultar_hors(cla,&hor_ini,&hor_fin);

     //*** obtenemos rangos horarios de preferencia de la clase

     arrcpy1(prefs,clases_fac[cla].pref_hors);
     dist_min=cant_hors;
     if ( prefs[0].hor_ini==prefs[0].hor_fin )
        dist_min=0;
     else
     {
      i=0;

      while((i<10) && (!( prefs[i].hor_ini==prefs[i].hor_fin)))
      {
       if ( hor_fin > prefs[i].hor_fin )
           dist= hor_fin-prefs[i].hor_fin;
       else
        if ( hor_ini < prefs[i].hor_ini )
           dist= prefs[i].hor_ini- hor_ini;
        else
           dist=0;
       if ( dist<dist_min )
          dist_min=dist;
       i++;
      }
     }
     total_dist+=dist_min;
    }
    return(total_dist);
}
```

```
int puedo_controlar_igual_hini(int car, int anio, int cur, int gru)
{

return(estructura_facu[car].anios_car[anio].cursos_anio[cur].grupos_cur[gru].
ctrl_eq_hini);
}



//****************** Controlar compactibilidad *********************

float controlar_compactibilidad(matriz2d asignacion)
{
    // cuenta la cantidad de 1/2 horas libres (puentes) para cada
    // salon, dia de la semana y turno.

    float costo;
    int sal, hor, cla;
    int dia_sem, tur, tur_fin, tur_ini;
    int encontre_cla, cant_agujeros;
    int hor_ini_real, hor_fin_real;

    costo=0;

    for(sal=0;sal<cant_salones;sal++)
        for (dia_sem=0; dia_sem<cant_dias_sem; dia_sem++)
            for(tur=0;tur<v_turnos.tope;tur++)
                {
                   encontre_cla = FALSE;
                cant_agujeros = 0;
                   tur_ini = v_turnos.vect[tur].h_ini;
                   tur_fin = v_turnos.vect[tur].h_fin;
                   hor_ini_real = tur_ini + dia_sem * cant_horas_dia;
                   hor_fin_real = tur_fin + dia_sem * cant_horas_dia;
                   for(hor = hor_ini_real; hor <= hor_fin_real; hor++)
                {
                   cla = asignacion.consultar(sal,hor);
                       if ( (cla!=(-1))&&(!encontre_cla) )
                       encontre_cla = TRUE;
                       if ( (cla==(-1))&&(encontre_cla) )
                              cant_agujeros++;
                       if ( (cla!=(-1))&&(encontre_cla)&&(cant_agujeros!=0) )
                   {
                          costo+= cant_agujeros;
                          cant_agujeros=0;
                   }
                   }

             }

    return(costo);
}




//************* Controlar adyacencia horaria entre   **************
//************* teoricos y practicos y que se dicten **************
//************* en el mismo salón                    **************

//************* Controlar igual hora de comienzo     **************
//************* entre los teoricos de un grupo       **************
```

```
float controlar_adyacencia_hor_teopra(vector_sh sh_clases,
                                                int car, int anio,
                                                int cur, int gru)
{

       int fin_clases_gru = FALSE;
       int cla = 0;
       int ind_cla, cant_cla_teo, cant_cla_pra;
        int cla_teo, cla_pra, sal_teo, sal_pra, i, j;
        int cla_teo_i, cla_teo_j, h_init_i, h_init_j, h_fint_i, h_fint_j ;
        int adyacentes, ady_sal;
        int h_inip, h_finp, h_init, h_fint, hdia_i, hdia_j;
       vector vec_teo(5);
       vector vec_pra(5);
      float costo=0;

       cant_cla_teo=cant_cla_pra=0;


        while(!fin_clases_gru)
      {
         ind_cla=
estructura_facu[car].anios_car[anio].cursos_anio[cur].grupos_cur[gru].clases_
gru[cla];
         if (ind_cla!=(-1))
        {
                  if(clases_fac[ind_cla].es_teorico)
                {
                     vec_teo.insertar(cant_cla_teo, ind_cla);
                   cant_cla_teo++;
                   }
                   else
                {
                     vec_pra.insertar(cant_cla_pra, ind_cla);
                     cant_cla_pra++;
                }

        }
         else
            fin_clases_gru=TRUE;
         cla++;
       }


      // Adyacencia en horario y salon entre teo y practicos

       if ((cant_cla_teo!=0)&&(cant_cla_pra!=0))
      {
          if (cant_cla_teo>=cant_cla_pra)
              // exijo que los practicos sean adyacentes a algun teo.
              for(i=0;i<cant_cla_pra;i++)
              {
               adyacentes = ady_sal = FALSE;
              cla_pra = vec_pra.consultar(i);
                  sh_clases.consultar_hors(cla_pra, &h_inip, &h_finp);
                  sal_pra = sh_clases.consultar_salon(cla_pra);
                  for(j=0;j<cant_cla_teo;j++)
                  {
                      cla_teo = vec_teo.consultar(j);
                      sh_clases.consultar_hors(cla_teo, &h_init, &h_fint);
                      if(((h_fint+1)==h_inip)||((h_finp+1)==h_init))
                   {
                          adyacentes=TRUE;
                          sal_teo = sh_clases.consultar_salon(cla_teo);
                          if(sal_pra==sal_teo)
                              ady_sal=TRUE;
```

```
                    break;
                }
            }
            if(!adyacentes)
         {
                costo++;
            }
            else
         {
                if(!ady_sal)
                    costo++;
            }
        }

     else
        // exijo que los teo. sean adyacentes a algun practico.
        for(i=0;i<cant_cla_teo;i++)
        {
        adyacentes=FALSE;
        cla_teo = vec_teo.consultar(i);
            sh_clases.consultar_hors(cla_teo, &h_init, &h_fint);
            sal_teo = sh_clases.consultar_salon(cla_teo);
            for(j=0;j<cant_cla_pra;j++)
            {
                cla_pra = vec_pra.consultar(j);
                sh_clases.consultar_hors(cla_pra, &h_inip, &h_finp);
                if(((h_fint+1)==h_inip)||((h_finp+1)==h_init))
             {
                    adyacentes=TRUE;
                    sal_pra = sh_clases.consultar_salon(cla_pra);
                    if(sal_pra==sal_teo)
                        ady_sal=TRUE;
                    break;
                }
            }
            if(!adyacentes)
         {
                costo++;
            }
            else
         {
                if(!ady_sal)
                    costo++;
            }

        }
 }


 //*********** Igual hora de comienzo en clases teoricas
 //***********       del mismo grupo


if (puedo_controlar_igual_hini(car, anio, cur, gru))
 {
     for(i=0;i<cant_cla_teo;i++)
   {
        cla_teo_i = vec_teo.consultar(i);
       sh_clases.consultar_hors(cla_teo_i, &h_init_i, &h_fint_i);

        for(j=i+1;j<cant_cla_teo;j++)
        {

            cla_teo_j = vec_teo.consultar(j);
            sh_clases.consultar_hors(cla_teo_j, &h_init_j, &h_fint_j);
            hdia_i=obtener_hora_del_dia(h_init_i);
```

```
                hdia_j=obtener_hora_del_dia(h_init_j);
                costo+=abs(hdia_i-hdia_j);
             }
           }

      }


      vec_teo.destr_vector();
      vec_pra.destr_vector();

       return(costo);

}




// ************* Calcular costo asignacion (funcion objetivo) *******
// ************* Aqui se controlan las preferencias

float hormiga::calcular_costo_asignacion()
{

    float costo, costo_esp, costo_ady, costo_total;
    int sal, hor, cla;
    int car, anio, cur, gru;

    costo_ady=0;
   costo_esp=0;
    costo=0;
    costo_total=0;

    float aux_rh, aux_cap, aux_desp, aux_oreq, aux_comp,aux_esp,aux_ady;


    costo = peso_capac_salon * controlar_capac_salon( sh_clases );
    costo_total+= costo;

   aux_cap=costo;
   //if (fmod(aux_iter, 10)==0)
    //fprintf(fcostos,"cap sal %.2f  ",costo);

    costo = peso_ran_hor * controlar_rangos_horarios( sh_clases );
    costo_total+= costo;

   aux_rh=costo;
    //if (fmod(aux_iter, 40)==0)
      fprintf(fcostos,"ranhor %.2f  ",costo);

    costo = peso_desper_salon * controlar_desper_salon( sh_clases );
    costo_total+= costo;

   aux_desp=costo;
    //if (fmod(aux_iter, 40)==0)
      fprintf (fcostos,"despsal %.2f  ",costo);

    costo = peso_otros_req * controlar_otros_req( sh_clases );
    costo_total+= costo;

   aux_oreq=costo;
    //if (fmod(aux_iter, 10)==0)
    //fprintf(fcostos,"otrosre %.2f  ",costo);

    costo = peso_compactibilidad * controlar_compactibilidad(asignacion);
```

```
    costo_total+= costo;

   aux_comp=costo;
    //if (fmod(aux_iter, 10)==0)
     fprintf(fcostos,"compact %.2f  ",costo);


   for(car=0;car<cant_carreras;car++)
    for(anio=0;anio<cant_anios;anio++)
      for(cur=0;cur<cant_cursos;cur++)
       for(gru=0;gru<cant_grupos;gru++)
       {
          costo_esp+= peso_espaciamiento * controlar_espaciamiento(sh_clases,
car, anio, cur, gru);
          costo_ady+= peso_adyacencia *
controlar_adyacencia_hor_teopra(sh_clases, car, anio, cur, gru);
       }

    costo_total+= costo_esp;

   aux_esp=costo_esp;
    //if (fmod(aux_iter, 10)==0)
        fprintf(fcostos,"espaciam %.2f  ",costo_esp);
    costo_total+= costo_ady;
    //if (fmod(aux_iter, 10)==0)
      fprintf(fcostos,"ady. hor %.2f  ",costo_ady);
   aux_ady=costo_ady;



    if (costo_total<costo_ant)
   {
      costo_rh=aux_rh;
      costo_cap=aux_cap;
      costo_desp=aux_desp;
      costo_oreq=aux_oreq;
      costo_comp=aux_comp;
      costo_espac=aux_esp;
      costo_adyac=aux_ady;
   }


    return (costo_total);
}
```

Matrices.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include "auxiliar.hpp"
#include "matrices.hpp"

extern float coef_evap;
extern int cant_horas_dia;
extern int cant_dias_sem;
extern float min_rastro;
extern int cant_cursos;
extern int cant_grupos;

extern FILE *salida;
extern FILE *frastros;



matriz3d::matriz3d()
{
    dim1=dim2=dim3=0;
}


matriz3d::matriz3d(int d1, int d2, int d3)
{
    int i ,j;

    matriz=(int***)calloc(d1,sizeof(int **));
    for(i=0;i<d1;i++)
    {
       matriz[i]=(int**)calloc(d2,sizeof(int*));
       for(j=0;j<d2;j++)
        matriz[i][j]=(int *)calloc(d3,sizeof(int));
     }

    if( matriz==NULL)
    {
      fprintf(stderr,"Error: No se pudo asignar memmoria para matriz 3d.\n");
       exit(-1);
     }

    dim1=d1;
    dim2=d2;
    dim3=d3;
}



int  matriz3d::consultar(int d1, int d2, int d3)
{
    return matriz[d1][d2][d3];
}



void matriz3d::insertar(int d1, int d2, int d3, int elem)
{
    matriz[d1][d2][d3]=elem;
}


void matriz3d::inicializar(int valor)
{
```

```
    int i, j, k;

   for(i=0;i<dim1;i++)
      for(j=0;j<dim2;j++)
         for(k=0;k<dim3;k++)
           matriz[i][j][k]=valor;

}


void matriz3d::acumular(int d1, int d2, int d3, int elem)
{
    matriz[d1][d2][d3]+=elem;
}



void matriz3d::mostrar()
{
    int i, j, k;


    for(k=0;k<dim3;k++)
   {
    fprintf(salida,"clase %i\n",k);
        for(i=0;i<dim1;i++)
        {
         for(j=0;j<dim2;j++)
            fprintf(salida,"%i  ",matriz[i][j][k]);
         fprintf(salida,"\n");
        }
   }
}



// ********************** Matriz 2d **********************

matriz2d::matriz2d()
{
    dim1=dim2=0;
}

matriz2d::matriz2d(int d1, int d2)
{
     int i ;

    matriz=(int**)calloc(d1,sizeof(int *));
    for(i=0;i<d1;i++)
     matriz[i]=(int*)calloc(d2,sizeof(int));

    if( matriz==NULL)
    {
      fprintf(stderr,"No se pudo asignar memmoria para matriz\n");
      exit(-1);
    }
    dim1=d1;
    dim2=d2;
}



int  matriz2d::consultar(int d1, int d2)
{
   return matriz[d1][d2];
}
```

```
void matriz2d::insertar(int d1, int d2,  int elem)
{
    matriz[d1][d2]=elem;
}


void matriz2d::inicializar()
{
   int i, j;

   for(i=0;i<dim1;i++)
    for(j=0;j<dim2;j++)
         matriz[i][j]=(-1);

}


void matriz2d::copiar(matriz2d mat)
{
       int sal, hor;

       for(sal=0;sal<dim1;sal++)
           for(hor=0;hor<dim2;hor++)
           matriz[sal][hor]=mat.matriz[sal][hor];
}


void matriz2d::consultar_dims(int *d1, int *d2)
{
   *d1=dim1;
   *d2=dim2;
}


void matriz2d::mostrar_dia(int dia)
{
   int s, h, hd;
   int cla;
   t_string10 id_clase;

   fprintf(salida,"\n****** dia %i ******\n",dia);

       for(s=0;s<dim1;s++)
       {
           for(hd=0; hd<cant_horas_dia; hd++)
               fprintf(salida,"I%03i        ",hd);
         fprintf(salida,"\n");

           for(hd=0; hd<cant_horas_dia; hd++)
       {
               h = dia * cant_horas_dia + hd;
           if (matriz[s][h]<0)
                   fprintf(salida,"nada        ");
               else
               {
                   obtener_id_clase(matriz[s][h], id_clase);
                   fprintf(salida,"%s   ",id_clase);
               }
           }
           fprintf(salida,"\n");

       }
```

```
}


void matriz2d::mostrar()
{
    int i,j;


    for(i=0;i<dim1;i++)
    {
        for(j=0;j<dim2;j++)
            fprintf(salida,"I%3i ",j);
        fprintf(salida,"\n");
        for(j=0;j<dim2;j++)
            fprintf(salida," %3i ",matriz[i][j]);
        fprintf(salida,"\n");
    }


    int dia;

    for(dia=0;dia<cant_dias_sem;dia++)
        mostrar_dia(dia);
}



// ********************** Vector ***************************

vector::vector()
{
    dim=0;
}

vector::vector(int d)
{

    vect=(int*)calloc(d,sizeof(int ));

    if( vect==NULL)
    {
      fprintf(stderr,"No se pudo asignar memoria para vector\n");
      exit(-1);
    }
    dim=d;
}



void vector::destr_vector()
{
       free(vect);
}


int  vector::consultar(int d)
{
    return vect[d];
}



void vector::insertar(int d, int elem)
{
    vect[d]=elem;
```

```
}


void vector::inicializar()
{
    int i;

    for(i=0;i<dim;i++)
      vect[i]=0;

}


int vector::consultar_dim()
{
    return (dim);
}



// ********************** Matriz 2d bool ********************

matriz2d_bool::matriz2d_bool()
{
    dim1=dim2=0;
}

matriz2d_bool::matriz2d_bool(int d1, int d2)
{
    int i ;

    matriz=(int**)calloc(d1,sizeof(int *));
    for(i=0;i<d1;i++)
     matriz[i]=(int*)calloc(d2,sizeof(int));

    if( matriz==NULL)
    {
      fprintf(stderr,"No se pudo asignar memmoria para matriz\n");
      exit(-1);
    }
    dim1=d1;
    dim2=d2;
}



int  matriz2d_bool::consultar(int d1, int d2)
{
    return matriz[d1][d2];
}



void matriz2d_bool::insertar(int d1, int d2)
{
    matriz[d1][d2]=TRUE;
}


void matriz2d_bool::inicializar()
{
    int i, j;

   for(i=0;i<dim1;i++)
    for(j=0;j<dim2;j++)
         matriz[i][j]=FALSE;
```

```
}



// ******************** matriz de asignacion ********************


mat_asignacion::mat_asignacion()
{
  dim1=dim2=0;
}


mat_asignacion::mat_asignacion(int d1, int d2)
{
    int i ;

    matriz=(t_string10**)calloc(d1,sizeof(t_string10 *));
    for(i=0;i<d1;i++)
     matriz[i]=(t_string10*)calloc(d2,sizeof(t_string10));

    if( matriz==NULL)
    {
      fprintf(stderr,"No se pudo asignar memoria para matriz asignacion\n");
      exit(-1);
    }
    dim1=d1;
    dim2=d2;
}


void mat_asignacion::consultar(int d1, int d2, t_string10 elem)
{
  strcpy(elem, matriz[d1][d2]);
}



void mat_asignacion::insertar(int d1, int d2, t_string10 elem)
{
  strcpy(matriz[d1][d2], elem);
}


void mat_asignacion::inicializar()
{
   int i, j;

   for(i=0;i<dim1;i++)
    for(j=0;j<dim2;j++)
         strcpy(matriz[i][j]," ");

}


// *************** vector tope ************************

vector_tope::vector_tope()
{}


vector_tope::vector_tope(int d):vector(d)
{
```

```
    tope=0;
}

int vector_tope::llena()
{
    if (tope==dim)
        return(TRUE);
    else
        return(FALSE);
}


void vector_tope::resetear()
{
    tope=0;
   inicializar();
}

void vector_tope::insertar(int elem)
{
    vect[elem]=TRUE;
    tope++;
}


int vector_tope::consultar_tope()
{
    return (tope);
}



// *************** vector tope entero ************************

vector_tope_entero::vector_tope_entero()
{}


vector_tope_entero::vector_tope_entero(int d):vector(d)
{
    tope=0;
}

int vector_tope_entero::llena()
{
    if (tope==dim)
        return(TRUE);
    else
        return(FALSE);
}


void vector_tope_entero::resetear()
{
    tope=0;
   inicializar();
}



void vector_tope_entero::insertar(int d, int elem)
{
     vect[d]=elem;
     tope++;
}
```

```
int vector_tope_entero::consultar_tope()
{
    return (tope);
}



// *************** matriz 2d indice ************************

matriz2d_indice::matriz2d_indice()
{}


matriz2d_indice::matriz2d_indice(int dim1, int dim2):
                      matriz2d_bool(dim1, dim2)
{
    i_fil=0;
   i_col=0;
}

int matriz2d_indice::llena()
{
    // Se considera llena si el indice de filas esta fuera del rango de la
matriz,
    // ya que de lo contrario se pierde la posibilidad de insertar en la
ultima
    // fila y columna.
    // Se considera solo el indice de filas dado que la matriz se recorre por
   // filas al actualizar o ajustar los indices.

    if (i_fil==dim1)
        return(TRUE);
    else
        return(FALSE);
}



int matriz2d_indice::asignado(int p_sal, int p_hor)
{
  return( matriz[p_sal][p_hor] );
}



void matriz2d_indice::resetear()
{
    i_fil=0;
   i_col=0;
   inicializar();
}

void matriz2d_indice::insertar(int fil, int col)
{
  int res;

  if(llena())
  {
       fprintf(stderr,"Error: Se intento insertar en matriz2d_indice y esta
llena\n");
       exit(-1);
  }
```

```
  matriz[fil][col]=TRUE;

  ajustar_indices(fil,col);
}


void matriz2d_indice::consultar_indices(int *fil, int *col)
{
    *fil = i_fil;
    *col = i_col;
}



void matriz2d_indice::act_indices()
{
    // se recorre la matriz por FILAS.

    if(llena())
    {
        fprintf(stderr,"Error: Se intento act. indices en matriz2d_indice y
esta llena\n");
        exit(-1);
    }


    if (i_col<dim2-1)
        i_col++;
    else
    {
        i_col=0;
        i_fil++;
    }

}



void matriz2d_indice::ajustar_indices(int sal, int hor)
{
    // se recorre la matriz por FILAS.

    if(llena())
    {
        fprintf(stderr,"Error: Se intento ajustar indices en matriz2d_indice y
esta llena\n");
        exit(-1);
    }


    if (hor<dim2-1)
        i_col=hor+1;
    else
    {
        i_col=0;
        i_fil=sal+1;
    }

}



//****************** matriz 3d real ******************************

matriz3d_real::matriz3d_real()
```

```
{
    dim1=dim2=dim3=0;
}

matriz3d_real::matriz3d_real(int d1, int d2, int d3)
{
     int i ,j;

    matriz=(float ***)calloc(d1,sizeof(float **));
    for(i=0;i<d1;i++)
    {
      matriz[i]=(float **)calloc(d2,sizeof(float *));
      for(j=0;j<d2;j++)
         matriz[i][j]=(float *)calloc(d3,sizeof(float));

    }
    if( matriz==NULL)
    {
      fprintf(stderr,"No se pudo asignar memoria para matriz 3d real\n");
      exit(-1);
    }
    dim1=d1;
    dim2=d2;
    dim3=d3;
}



float matriz3d_real::consultar(int d1, int d2, int d3)
{
    if ((d1<0)||(d2<0)||(d3<0))
    {
       fprintf(stderr,"Indice negativo en matriz3d_real::consultar\n");
       exit(-1);
    }

    return matriz[d1][d2][d3];
}



void matriz3d_real::insertar(int d1, int d2, int d3, float elem)
{
    if ((d1<0)||(d2<0)||(d3<0))
    {
       fprintf(stderr,"Indice negativo en matriz3d_real::insertar\n");
       exit(-1);
    }

    matriz[d1][d2][d3]=elem;
  }

  void matriz3d_real::inicializar(float valor)
  {
    int i, j, k;

   for(i=0;i<dim1;i++)
      for(j=0;j<dim2;j++)
         for(k=0;k<dim3;k++)
            matriz[i][j][k]=valor;

}


void matriz3d_real::acumular(int d1, int d2, int d3, float elem)
```

```
{
    if ((d1<0)||(d2<0)||(d3<0))
   {
       fprintf(stderr,"Indice negativo en matriz3d_real::acumular\n");
       exit(-1);
   }

    matriz[d1][d2][d3]+=elem;
}


void matriz3d_real::actualizar(matriz3d_real mat)
{
    int i, j, k;

   for(i=0;i<dim1;i++)
      for(j=0;j<dim2;j++)
         for(k=0;k<dim3;k++)
        {
            matriz[i][j][k]=(matriz[i][j][k]*coef_evap)+mat.consultar(i,j,k);
            if (matriz[i][j][k] < min_rastro)
                matriz[i][j][k] = min_rastro;
         }
}


void matriz3d_real::mostrar()
{
    int i, j, k;

    for(k=0;k<dim3;k++)
    {
    fprintf(frastros,"clase %i\n",k);
        for(i=0;i<dim1;i++)
        {
         for(j=0;j<dim2;j++)
            fprintf(frastros,"%.2f  ",matriz[i][j][k]);
         fprintf(frastros,"\n");
        }
    }
}



// ******** elemento sh_clase del vector_sh:
// Consta de un salón, un horario inicial y un horario final.

sh_clase::sh_clase()
{
    h_ini = h_fin = salon = 0;
}

sh_clase::sh_clase(int sal, int hor_ini, int hor_fin)
{
    salon = sal;
    h_ini = hor_ini;
    h_fin = hor_fin;
}

int sh_clase::consultar_sal()
{
    return(salon);
}


void sh_clase::consultar_hors(int *hor_ini, int *hor_fin)
```

```
{
    *hor_ini=h_ini;
    *hor_fin=h_fin;
}


//*****************vector sh*****************************

vector_sh::vector_sh()
{
    dim=0;
}


vector_sh::vector_sh(int d)
{
    vec=(sh_clase *)calloc(d,sizeof(sh_clase));
    if( vec==NULL)
    {
        fprintf(stderr,"Error: No se pudo asignar memoria para vector_sh.\n");
       exit(-1);
    }
    dim=d;
}



void vector_sh::inicializar()
{
    int i;
    for(i=0;i<dim;i++)
    {
        insertar(i,0,0,0);
    }
}

int vector_sh::consultar_dim()
{
    return(dim);
}



int vector_sh::consultar_salon(int cla)
{
    return(vec[cla].consultar_sal());
}


void vector_sh::consultar_hors(int cla, int *hor_ini, int *hor_fin)
{
    vec[cla].consultar_hors(hor_ini, hor_fin);
}


void vector_sh::insertar(int cla ,int sal, int hor_ini, int hor_fin)
{
    vec[cla]=sh_clase(sal, hor_ini, hor_fin);
}


// *************** vector ids grupo ************************
// vector de identificadores de grupos.

vector_ids_grupo::vector_ids_grupo()
{}
```

```
vector_ids_grupo::vector_ids_grupo(int d)
{
    vect=(t_string5 *)calloc(d,sizeof(t_string5));

   dim=d;
    tope=0;
}



int vector_ids_grupo::pertenece( t_string5 elem )
{
  int i;
  int encontre;

  encontre=FALSE;
  i=0;
  while( (i<tope) && (!encontre))
  {
      if (cmp_str(vect[i],elem,5))
          encontre=TRUE;
     else
          i++;
  }

  return(encontre);
}



void vector_ids_grupo::insertar( t_string5 elem)
{
    if (tope==dim)
    {
    fprintf(stderr,"Error: se intento insertar en\n");
        fprintf(stderr,"vector_ids_grupo pero esta lleno\n");
        exit(-1);
   }
    strcpy(vect[tope],elem);
    tope++;
}



int vector_ids_grupo::consultar_tope()
{
    return (tope);
}


void vector_ids_grupo::inicializar()
{
    tope=0;
}

void vector_ids_grupo::mostrar()
{
    int i;

    for(i=0;i<tope;i++)
        fprintf(salida,"%s  ",vect[i]);
}

// ******************* matriz horscaranio *******************
```

```
mat_horscaranio::mat_horscaranio()
{
  dim1=dim2=dim3=0;
}


mat_horscaranio::mat_horscaranio(int d1, int d2, int d3)
{
    int i, j, k;

    matriz=(vector_ids_grupo ***)calloc(d1,sizeof(vector_ids_grupo **));
     for(i=0;i<d1;i++)
     {
      matriz[i]=(vector_ids_grupo **)calloc(d2,sizeof(vector_ids_grupo *));
     for(j=0;j<d2;j++)
        matriz[i][j]=(vector_ids_grupo *)calloc(d3,sizeof(vector_ids_grupo));
    }


    if( matriz==NULL)
    {
       fprintf(stderr,"No se pudo asignar memoria para matriz
hors_caranio\n");
      exit(-1);
    }
    dim1=d1;
     dim2=d2;
     dim3=d3;

     for(i=0;i<dim1;i++)
     for(j=0;j<dim2;j++)
      for(k=0;k<dim3;k++)
          matriz[i][j][k]=vector_ids_grupo(cant_cursos * cant_grupos);

}



int mat_horscaranio::pertenece(int d1, int d2, int d3, t_string5 elem)
{
  // indica si el elemento esta o no en horscaranio.


    if ((d1<0)||(d2<0)||(d3<0))
    {
        fprintf(stderr,"Indice negativo en mat_hors_caranio::consultar\n");
        exit(-1);
    }

    return(matriz[d1][d2][d3].pertenece(elem));

}


void mat_horscaranio::insertar(int d1, int d2, int d3, t_string5 elem)
{
    vector_ids_grupo p_ids_grupo;

    if ((d1<0)||(d2<0)||(d3<0))
    {
        fprintf(stderr,"Indice negativo en mat_hors_caranio::insertar\n");
        exit(-1);
    }
```

```
    matriz[d1][d2][d3].insertar(elem);

}


void mat_horscaranio::inicializar()
{
    // Inicializa todos los vectores de ids_grupos (tope = 0),
   // que son los elementos de mat_horscaranio.

    int i, j, k;

    for(i=0;i<dim1;i++)
     for(j=0;j<dim2;j++)
      for(k=0;k<dim3;k++)
          matriz[i][j][k].inicializar();

}


void mat_horscaranio::mostrar()
{
    int i, j, k;

    for(i=0;i<dim1;i++)
   {
    fprintf(salida,"carrera %i\n",i);
        for(j=0;j<dim2;j++)
        {
         fprintf(salida,"año %i\n",j);
         for(k=0;k<dim3;k++)
         {
            fprintf(salida,"k: %i  ",k);
            matriz[i][j][k].mostrar();
          fprintf(salida,"\n");
        }
         fprintf(salida,"\n");
        }
   }
}




// ********************** elemento de matriz 2d grupos ************
// el elemento consta de un identificador de clase y un salon.

t_elem_cs::t_elem_cs()
{
    strcpy(id_clase," ");
    strcpy(id_salon," ");
}

t_elem_cs::t_elem_cs( t_string10 p_id_clase, char p_id_salon[4] )
{
    strcpy(id_clase, p_id_clase);
    strcpy(id_salon, p_id_salon);
}

void t_elem_cs::insertar( t_string10 p_id_clase, char p_id_salon[4])
{
    strcpy(id_clase, p_id_clase);
    strcpy(id_salon, p_id_salon);
}
```

```
void t_elem_cs::cons_id_clase(t_string10 p_id_clase)
{
    strcpy(p_id_clase, id_clase);
}

void t_elem_cs::cons_id_salon(char p_id_salon[4])
{
    strcpy(p_id_salon, id_salon);
}



// ********************** Matriz 2d grupos ********************
// matriz para visualizar la asignacion por grupos.

matriz2d_gru::matriz2d_gru()
{
    dim1=dim2=0;
}

matriz2d_gru::matriz2d_gru(int d1, int d2)
{
     int i ;

    matriz=(t_elem_cs**)calloc(d1,sizeof(t_elem_cs*));
    for(i=0;i<d1;i++)
       matriz[i]=(t_elem_cs*)calloc(d2,sizeof(t_elem_cs));

    if( matriz==NULL)
    {
      fprintf(stderr,"No se pudo asignar memoria para matriz2d_gru\n");
      exit(-1);
    }
    dim1=d1;
    dim2=d2;
}


void matriz2d_gru::consultar(int d1, int d2, t_elem_cs &elem)
{
     elem = matriz[d1][d2];
}



void matriz2d_gru::insertar(int d1, int d2,  t_elem_cs elem)
{
    matriz[d1][d2] = elem;
}



void matriz2d_gru::inicializar()
{
    int i, j;
    t_elem_cs elem_cs("         ", "   ");

    for(i=0;i<dim1;i++)
     for(j=0;j<dim2;j++)
        insertar(i, j, elem_cs);

}



void matriz2d_gru::consultar_dims(int *d1, int *d2)
```

```
{
    *d1=dim1;
    *d2=dim2;
}



void matriz2d_gru::mostrar()
{
    int i,j;
    t_elem_cs elem_cs;
    t_string10 id_clase;
    char id_salon[4];


        fprintf(salida,"\n     ");
        for(j=0;j<dim2;j++)
            fprintf(salida,"dia %1i       ",j);
        fprintf(salida,"\n");

        for(i=0;i<dim1;i++)
        {

            fprintf(salida,"h%2i  ", i);
            for(j=0;j<dim2;j++)
            {
                consultar(i, j, elem_cs);
                elem_cs.cons_id_clase(id_clase);
                fprintf(salida,"%s   ",id_clase);
            }
            fprintf(salida,"\n     ");

            for(j=0;j<dim2;j++)
            {
                consultar(i, j, elem_cs);
                elem_cs.cons_id_salon(id_salon);
                fprintf(salida,"%s         ",id_salon);
            }
          fprintf(salida,"\n");

        }


}



//******************* matriz 3d str ***********************************
// matriz de 3 dimensiones cuyos elementos son strings de 5 caracteres


matriz3d_str::matriz3d_str()
{
    dim1=dim2=dim3=0;
}

matriz3d_str::matriz3d_str(int d1, int d2, int d3)
{
    int i ,j;

   matriz=(t_string5 ***)calloc(d1,sizeof(t_string5 **));
   for(i=0;i<d1;i++)
   {
      matriz[i]=(t_string5 **)calloc(d2,sizeof(t_string5 *));
      for(j=0;j<d2;j++)
```

```
         matriz[i][j]=(t_string5 *)calloc(d3,sizeof(t_string5));

   }
   if( matriz==NULL)
   {
       fprintf(stderr,"No se pudo asignar memoria para matriz 3d str\n");
      exit(-1);
   }
   dim1=d1;
    dim2=d2;
    dim3=d3;
}



void matriz3d_str::consultar(int d1, int d2, int d3, t_string5 elem)
{
    if ((d1<0)||(d2<0)||(d3<0))
   {
       fprintf(stderr,"Indice negativo en matriz3d_str::consultar\n");
       exit(-1);
   }

    strcpy(elem, matriz[d1][d2][d3]);
}



void matriz3d_str::insertar(int d1, int d2, int d3, t_string5 elem)
{
    if ((d1<0)||(d2<0)||(d3<0))
   {
       fprintf(stderr,"Indice negativo en matriz3d_str::insertar\n");
       exit(-1);
   }

   strcpy(matriz[d1][d2][d3],elem);
}


void matriz3d_str::inicializar()
{
    int i, j, k;

   for(i=0;i<dim1;i++)
      for(j=0;j<dim2;j++)
         for(k=0;k<dim3;k++)
            strcpy(matriz[i][j][k]," ");

}


void matriz3d_str::mostrar()
{
    int i, j, k;


    for(i=0;i<dim1;i++)
    {
        fprintf(salida,"carrera: %2i\n",i);
        for(j=0;j<dim2;j++)
        {
            fprintf(salida,"año: %2i  grupos:  ",j);
            for(k=0;k<dim3;k++)
                fprintf(salida,"%s  ",matriz[i][j][k]);
            fprintf(salida,"\n");
```

```
        }
    }
    fprintf(salida,"\n");
}
```

Princip.cpp

```
#include <time.h>
#include <math.h>
#include "auxiliar.hpp"
#include "matrices.hpp"
#include "hormiga.hpp"
#include "estrfacu.hpp"
#include "cargadat.hpp"
#include "princip.hpp"

#define VACIO -1


// nuevas variables para funciones de visibilidad y rastros:
float costo_max, costo_min, gama;
float costo_vis;

float max_cvis, c1;


// Parámetros para la estructura de la facultad:

int cant_clases;
int cant_hors;
int cant_salones;
int cant_clases_gru;
int cant_grupos;
int cant_cursos;
int cant_anios;
int cant_carreras;
int cant_horas_dia;
int cant_dias_sem;

// Parámetros del algoritmo:

int cant_horm;
int max_iter;
float exp_rastro;
float exp_visib;
float coef_evap;
float max_costo;
float mult_rastro;
float min_visib;
float max_visib;
float min_prob;
float min_rastro;

// Pesos para cálculo de preferencias:

float peso_ran_hor;
float peso_capac_salon;
float peso_otros_req;
float peso_desper_salon;
float peso_espaciamiento;
float peso_adyacencia;
float peso_compactibilidad;

// Estructura de la facultad:

t_carreras_fac estructura_facu;
t_vect_turnos v_turnos;
```

```
t_salones salones_fac;
t_clases_fac clases_fac; // vector con todas las clases de facultad.
t_perfiles perfiles;


// Archivos para controlar resultados:

FILE *salida;
FILE *frastros;
FILE *fcostos;

int aux_iter=0; // para debuggear
float costo_ant; // se declara global para debuggear
float costo_rh, costo_cap, costo_desp;     //son para debuggear
float costo_oreq, costo_comp, costo_espac,costo_adyac;//son para debuggear



//******** Fin. Determina si el algorimo debe detenerse ********

int fin(int cant_iter)
{
    if (cant_iter > max_iter)
        return(TRUE);

    return(FALSE);

}

// ********** insertar clases de un grupo en la matriz mat_gru *******

void insertar_clases_grupo_en_mat_gru(int car, int anio, int cur, int gru,
                                            vector_sh   sol_sh_clase,
                                            matriz2d_gru &mat_gru)
{
    int fin_clases_gru;
    int cla, ind_cla;
    t_string10 id_clase;
   char id_salon[4];
    int sal, hor_ini, hor_fin, dia, h, h_dia;
    t_elem_cs elem_cs(" ", " ");
    t_elem_cs ant_elem_cs(" ", " ");


    fin_clases_gru=FALSE;
    cla=0;
    while(!fin_clases_gru)
   {
        ind_cla=
estructura_facu[car].anios_car[anio].cursos_anio[cur].grupos_cur[gru].clases_
gru[cla];
        if (ind_cla!=(-1))
    {
            obtener_id_clase(ind_cla, id_clase);
            sal = sol_sh_clase.consultar_salon(ind_cla);
            obtener_id_salon(sal, id_salon);
            elem_cs.insertar(id_clase, id_salon);
        sol_sh_clase.consultar_hors(ind_cla, &hor_ini, &hor_fin);
            dia = obtener_dia(hor_ini);
            for(h=hor_ini; h<=hor_fin; h++)
        {
                h_dia = obtener_hora_del_dia(h);

                mat_gru.consultar(h_dia, dia, ant_elem_cs);
```

```
                ant_elem_cs.cons_id_clase(id_clase);

                if(!cmp_str(id_clase,"  ",1))
            {
                    fprintf(stderr,"Error: se intento insertar en
mat_gru[%i][%i] y\n",
                                h_dia, dia);
                    fprintf(stderr,"ya estaba ocupado por: %s\n", id_clase);
                    exit(-1);
                }

                mat_gru.insertar(h_dia, dia, elem_cs);
            }
        }
        else
            fin_clases_gru=TRUE;
        cla++;
    } // while

}



// ********************** mostrar asignacion por grupos **************

void mostrar_asignacion_por_grupos( vector_sh sol_sh_clase,
                                            matriz3d_str gru_caranio )
{
    matriz2d_gru mat_gru(cant_horas_dia, cant_dias_sem);
    int car, anio, cur, gru;

    // variables para gru_caranio:
    int gru_ca; // indice.
    t_string5 id_gru_ca; // identificador de grupo.
    int fin_grupos_ca; // indica fin de los grupos de una carrera-año.

    char id_carrera[4];
    char id_anio[3];
    char id_curso[5];
    char id_grupo[5];


    fprintf(salida,"\n*******************************\n");
    fprintf(salida,"**** asignacion por grupos ****\n");
    fprintf(salida,"*******************************\n");


    // recorro estructura de la facultad y cargo matriz de grupo con
    // las clases de un grupo:

    for(car=0;car<cant_carreras;car++)
    {
        strcpy(id_carrera, estructura_facu[car].id_carrera);
        for(anio=0;anio<cant_anios;anio++)
        {
            strcpy(id_anio, estructura_facu[car].anios_car[anio].id_anio);
            gru_ca = 0;
          fin_grupos_ca = FALSE;
            while(!fin_grupos_ca)
            {
                gru_caranio.consultar(car, anio, gru_ca, id_gru_ca);
                if ( strlen(id_gru_ca)==0 )
                    fin_grupos_ca = TRUE;
                else
                {
                    mat_gru.inicializar();
```

```
                    for(cur=0;cur<cant_cursos;cur++)
                        for(gru=0;gru<cant_grupos;gru++)
                        {
                            strcpy(id_grupo,
estructura_facu[car].anios_car[anio].cursos_anio[cur].grupos_cur[gru].id_grup
o);
                            if (cmp_str(id_grupo, " ", 1))
                                break;
                            if(cmp_str(id_grupo, id_gru_ca, 4)) // es uno de los
grupos
                                // correspondientes al grupo que estoy mostrando
ahora
                                insertar_clases_grupo_en_mat_gru(car, anio, cur,
gru,

    sol_sh_clase, mat_gru);
                            } // for grupos
                  } // else fin_grupos_ca

                  // mostrar la planilla de clases de un grupo:
                  if(!fin_grupos_ca)
                {
                      fprintf(salida,"\n****** car: %s año: %s gru: %s\n",
                          id_carrera, id_anio, id_gru_ca);
                      mat_gru.mostrar();
                }

                gru_ca++;
              } // while grupos_caranio
            } // for anios
        } // for carreras

}

// *************** Mostrar resultados *****************

void mostrar_resultados(int encontro_sol, matriz2d solucion,
                            matriz3d_real mat_rastros, int iter_min_costo,
                            vector_sh sol_sh_cla, matriz3d_str gru_caranio)
{
    if(encontro_sol)
    {
        fprintf(salida,"\n\nSOLUCION FINAL \n");
        solucion.mostrar();
        fprintf(salida,"El costo de la mejor solución es: %.2f \n",costo_ant);
        fprintf(salida,"costo_rh es : %.4f \n", costo_rh);
        fprintf(salida,"costo_cap es : %.4f \n",costo_cap);
        fprintf(salida,"costo_desp es : %.4f \n", costo_desp);
        fprintf(salida,"costo_oreq es : %.4f \n", costo_oreq);
        fprintf(salida,"costo_comp es : %.4f \n",costo_comp);
        fprintf(salida,"costo_espac es : %.4f \n",costo_espac);
        fprintf(salida,"costo_ady es : %.4f \n", costo_adyac);

    fprintf(salida,"Iteracion de la mejor solución : %i \n",iter_min_costo);

        mostrar_asignacion_por_grupos(sol_sh_cla, gru_caranio);
    }
    else
    {
        fprintf(salida,"\n*****************************\n");
        fprintf(salida,"*** no se encontro solucion ***\n");
        fprintf(salida,"*****************************\n");

        printf("\n*****************************\n");
        printf("*** no se encontro solucion ***\n");
        printf("*****************************\n");
```

```
   }


   fprintf(frastros,"mat. rastros definitiva\n");
     mat_rastros.mostrar();
}



//*********************************************************************
//**********************   M  A  I  N   ******************************
//*********************************************************************


main( int argc, char *argv[] )
{
    // han declarado como globales los datos de la
    // estrucura de la facultad.

    int m;

    if (argc !=2)
    {
     fprintf(stderr,"Sintaxis de invocacion:carga nom_archivo");
     exit(-1);
    }

    salida = fopen("salida.txt","w");
   frastros = fopen("frastros.txt","w");
    fcostos = fopen("fcostos.txt","w");


    carga_de_dimensiones(argv[1]);

    matriz3d_real mat_rastros(cant_salones,cant_hors,cant_clases);

    // matriz con los grupos pertenecientes a una carrera y año.
    // se utiliza para desplegar la asignacion por grupos:
    matriz3d_str gru_caranio(cant_carreras, cant_anios,
                             cant_cursos * cant_grupos);

    carga_de_datos(argv[1], gru_caranio);


    fprintf(salida, "\nen main\n");

    fprintf(salida, "cant_carr %i\n",cant_carreras);
    fprintf(salida, "cant_anios %i\n",cant_anios);
    fprintf(salida, "cant_cursos %i\n",cant_cursos);
    fprintf(salida, "cant_grupos %i\n",cant_grupos);
    fprintf(salida, "cant_clases_gru %i\n",cant_clases_gru);
    fprintf(salida, "cant_clases %i\n",cant_clases);
    fprintf(salida, "cant_salones %i\n",cant_salones);




    // ********* Algoritmo principal: **********



    matriz3d_real mat_rastros_aux(cant_salones, cant_hors, cant_clases);
    matriz2d      solucion(cant_salones, cant_hors);
```

```
// para debuggear: solucion_trunc:
matriz2d      solucion_trunc(cant_salones, cant_hors);
vector_sh     sol_sh_cla(cant_clases);

hormiga coquita(cant_salones,cant_hors,cant_clases);

float costo;

//float costo_ant; // se usa global para debuggear

time_t hora;
int cant_iter;
int movio; // indica si la hormiga pudo asignar una clase
            // de lo contrario paso a la siguiente hormiga.

int encontro_sol; // indica si el algoritmo encontro al menos
                      // una solucion.
int encontro_sol_iter; // indica si el algoritmo encontro al
                           // menos una solucion en la iteracion actual.

int i, j, k ;
t_string10 elem, cla;
int iter_min_costo;


//variables para nuevas funciones de
//visibilidad y rastros:

costo_max=150;
costo_min=5;
gama=2.0;
costo_vis=0;

max_cvis=0;
c1=100;

// vin nuevas variables.


costo_ant=RAND_MAX;


coquita.resetear();


srand((unsigned) time(&hora));

cant_iter=0;

mat_rastros.inicializar(min_rastro);

encontro_sol = FALSE;


do
{
    mat_rastros_aux.inicializar(0);
    encontro_sol_iter = FALSE;

    for (i=1;i<=cant_horm;i++)  // para cada hormiga
    {
        do
        {
          movio = coquita.mover(mat_rastros);
          aux_iter++;  // para debuggear.
        }
```

```
       while ((!coquita.llena_lt())&&(movio));

        if (movio)
        {
        encontro_sol = TRUE;
            encontro_sol_iter = TRUE;

            costo = coquita.calcular_costo_asignacion();

            //if (fmod(aux_iter, 40)==0)
            //{

            fprintf(fcostos,"iteracion: %i  ",cant_iter);
                fprintf(fcostos,"solucion: %.2f\n", costo);
            //}

            if (costo < costo_ant)
            {
                coquita.copiar_asig(solucion);
                coquita.copiar_sh_clase(sol_sh_cla);
                costo_ant = costo;
             iter_min_costo = cant_iter;
            }


            // almacenar rastro en matriz temporal:

            coquita.dejar_rastro(costo, mat_rastros_aux);

           //  fprintf(salida,"mat. rastros auxiliar\n");
           //  mat_rastros_aux.mostrar();
        }
        else  // para debuggear
       {
            coquita.copiar_asig(solucion_trunc);
         // fprintf(salida,"**** solucion truncada:\n");
           //   solucion_trunc.mostrar();
        }

        coquita.resetear();  // lista tabu, lista sh, asignacion.

    } // para cada hormiga

    // agregar rastros dejados por las hormigas a la matriz de rastros y
    // actualizarla segun el coeficiente de evaporacion:

    if (encontro_sol_iter)
    {
        mat_rastros.actualizar(mat_rastros_aux);
    }

    if (fmod(cant_iter, 100)==0)
   {
    fprintf(frastros,"mat. rastros definitiva en iter: %i\n",cant_iter);
        mat_rastros.mostrar();
    }


    cant_iter++;
    if (fmod(cant_iter, 10)==0)
        printf("Iteracion : %i\n",cant_iter);


}
 while (!fin(cant_iter));
```

```
    mostrar_resultados(encontro_sol, solucion, mat_rastros,
                           iter_min_costo, sol_sh_cla, gru_caranio);

    fclose(frastros);
    fclose(salida);
    fclose(fcostos);

    printf("Terminé. Andá a leer el archivo \n");

    return 0;


}
```