ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

INSTITUTO DE CIENCIAS MATEMÁTICAS
ESCUELA DE GRADUADOS

TESIS DE GRADUACIÓN

PREVIO A LA OBTENCIÓN DEL TÍTULO DE
MAGÍSTER EN CONTROL DE OPERACIONES Y GESTIÓN LOGÍSTICA

TEMA

DISEÑO E IMPLEMENTACIÓN DE UNA HEURÍSTICA PARA RESOLVER EL PROBLEMA DE CALENDARIZACIÓN DE HORARIOS PARA UNIVERSIDADES

AUTOR
JOSE XAVIER CABEZAS GARCÍA

Guayaquil-Ecuador
2009

Dedicatoria

A mi esposa Lena y a mis hijos Daniela y Xavier Eduardo. Los amo con todas mis fuerzas.

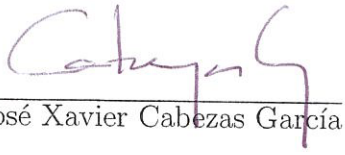
Agradecimientos

A Dios por darme la fortaleza para alcanzar mis metas, a mis padres por su apoyo incondicional, al Mat. Fernando Sandoya por su acertada dirección en esta investigación y a mis compañeros de trabajo Mat. Johnni Bustamante e Ing. Erwin Delgado por su amistad y por sus útiles consejos profesionales.

Declaración Expresa

La responsabilidad del contenido de esta Tesis de Postgrado, me corresponde exclusivamente; y el patrimonio intelectual del mismo al **ICM (Instituto de Ciencias Matemáticas)** de la ESPOL (Escuela Superior Politécnica Del Litoral).

(Reglamento de Graduación de la ESPOL)



José Xavier Cabezas García

TRIBUNAL DE GRADUACIÓN

M.Sc. Félix Ramírez Cruz
PRESIDENTE DEL
TRIBUNAL

M.Sc. Fernando Sandoya Sanchez
DIRECTOR DE TESIS

M.Sc. Johnny Bustamante Romero
VOCAL

Índice general

Índice de figuras

Índice de tablas

Índice de funciones y procedimientos en Matlab®

Capítulo 1

El Problema de Calendarización de Horarios para Universidades

1.1. Introducción a la Calendarización

El *Problema de Calendarización* **TTP** (por sus siglas en inglés *Timetabling*) está dentro de un conjunto de problemas más amplio denominados simplemente de *Programación* **(Scheduling)** [11] y se puede definir de forma muy simple. A continuación un par de definiciones que aparecen en la literatura:

Definición 1.1.1 (Calendarización, Zhipeng Lu and Jin-Kao Hao [7])**.** *Asignar un número de eventos, cada uno con ciertas características, a un número limitado de recursos sujeto a restricciones.*

Lance D. Chambers [2] define también al **TTP** de una forma más general:

Definición 1.1.2 (Calendarización, Chambers L.)**.** *El Timetabling (Calendarización) se puede definir como aquello que describe, dónde y cuándo las personas y los recursos deben estar en un instante dado.*

Para poder interpretar lo que en la definición 1.1.1 significan los términos: eventos, recursos y restricciones, se debe estudiar las características de un problema particular. Para este trabajo, se considera una clase de **TTP's** denominada *Educacional*, que consiste en la programación de *clases* de un conjunto de *materias* con un número dado de *aulas* y *periodos de tiempo* disponibles que satisfacen varias restricciones [1], ver Figura 1.1. Las restricciones pueden ser clasificadas en *duras* y *suaves*. Las restricciones de tipo *duras* son utilizadas para encontrar soluciones factibles al problema y no pueden ser infringidas, por otro lado las restricciones *suaves* pueden relajarse de tal forma de buscar soluciones cercanas al óptimo que serán cotas (inferiores o superiores) para buscar nuevas alternativas de solución.

El caso *Educacional* es solo una de las variaciones de los **TTP's**, también existen otras aplicaciones en áreas muy distintas, tales como transporte (programación de salidas de buses), TV (Calendario de Programas de TV) o deportes (programación de calendarios de juegos), un ejemplo de éste último tipo de problema se encuentra en [15], donde se muestra un método de solución que utiliza una aproximación con

		Sistema de Planificación Académica		1er. Término 2009	Profesor: Xavier Cabezas	
		Lunes	Martes	Miércoles	Jueves	Viernes
7:30	8:30	Logística 1 - Par 1 BA18		Logística 1 - Par 1 BA18		
8:30	9:30					
9:30	10:30					Diseño de Experimentos (IAPI)-Par 2 COMP B
10:30	11:30					
11:30	12:30	Procesos Estocásticos - Par 1 ED25		Procesos Estocásticos - Par 1 ED25		
12:30	13:30					

Figura 1.1: Ejemplo de Calendario Eduacional (Timetable)

programación lineal entera (**ILP**, por las siglas en inglés de *Integer Linear Programming*) para la determinación de horarios de la Liga Italiana de Fútbol.

Para el caso *Educacional* se pueden clasificar en dos grupos: *Calendarización para Exámenes* (**ETT**, *Exam Timetabling*) y *Calendarización para Materias* (*Clases Regulares*) (*CTT*, *Course Timetabling*), este último a su vez puede ser subdividido en *Calendarización por Materias Basado en Inscripciones* (**EB-CTT**, *Enrollment-Based Course Timetabling*) y *Calendarización por Materias Basado en Plan de Estudios* (**CB-CTT**, *Curriculum-Based Course Timetabling*), ver Tabla 1.1.

Tabla 1.1: Clasificación de Problemas de Calendarización Educacional

Calendarización		
Educacional	Examenes **ETT**	
	Materias **CTT**	Basado en Inscripciones **EB-CTT**
	Materias **CTT**	Basado en Plan de Estudios **CB-CTT**

Cuando se intenta resolver el **TTP**, pueden presentarse una serie de *conflictos* entre las restricciones que se plantean, pero éstas para el caso del **EB-CTT** se deberán a la decisión que tomen los estudiantes sobre las materias que desean tomar, un cruce podría ocurrir si la elección no ha sido buena; por otro lado los problemas que se podrán presentar en el **CB-CTT** resultarán de una programación no tan buena del plan de estudios por parte de la institución educativa.

Los *conflictos* ocurren cuando una programación requiera que cualquiera de los recursos esté en dos lugares al mismo tiempo. La *restricciones de equipo* varían evidentemente entre instituciones, entre exámenes y programaciones de clases regulares. En realidad todo depende del tamaño de la institución o del departamento que ofrece sus cursos y del tiempo requerido para producir la solución.

Otra de las cosas complicadas en los problemas de *Calendarización* es el lograr satisfacer las restricciones de las diferentes personas que tienen interés sobre los resultados obtenidos. Los principales *stakeholders*[1] en este problema con sus princi-

[1]Stakeholders es un término anglosajón utilizado para referirse a personas que tienen interés en un proceso determinado.

pales objetivos y necesidades son:

1. **La Administración:** La cual define el mínimo estándar requerido para una *"Tabla de Tiempo"*, como por ejemplo, el número de exámenes que un estudiante debería tomar en un día específico.

2. **Los Departamentos:** En algunas ocasiones, los departamentos dentro de las instituciones o institutos suelen tener demandas específicas sobre aulas de clases o laboratorios.

3. **Estudiantes:** Es muy complicado obtener una *tabla de tiempo* que satisfaga todas las necesidades de cada uno de los estudiantes. Algunos estudiantes prefieren no tener clases regulares muy tarde y otros en cambio lo prefieren así para tener oportunidad de trabajar. Por otro lado, una exigencia muy común, es que entre cada examen consecutivo exista un descanso (*break*), que les premita un pequeño espacio entre prueba y prueba o entre clase y clase.

Los profesores Burke, Jackson, Kingston y Weare [10] mencionan que entre las diferentes formas de los problemas de *Calendarización* se pueden encontrar restricciones comunes que los relacionan unos con otros:

- **Asignación de recursos:** Cuando se requieren diferentes recursos dependiendo de las necesidades de los cursos dictados o exámenes a tomar.

- **Asignación de tiempo:** Esta restricción se utiliza cuando se desea días para los cuales los profesores están disponibles, o para pre asignar tiempo a una clase o examen particular.

- **Restricción de tiempo entre clases o exámenes:** Estas restricciones se refieren a las condiciones de tiempo entre sesiones, por ejemplo, un conjunto de exámenes podrían querer tomarse al mismo tiempo, o un conjunto de clases podrían querer ser dadas en un orden determinado.

- **Sesiones dispersas:** Las sesiones deberían ser dispersas en el tiempo. Este tipo de restricciones son las mencionadas anteriormente y que son impuestas por la administración, por ejemplo, podría requerirse que los estudiantes no tomen más de un examen en un día y si lo hacen, deberían ser muy pocos, para que hayan intervalos adecuados de descanso.

- **Sesiones coherentes:** Estas restricciones son las que consideran condiciones particulares de los participantes. Las restricciones de los estudiantes forman parte de este juego, porque deberían exigir intervalos de descanso entre exámenes si tienen que tomar más de una prueba en un día, o los profesores quizás prefieran dar clases regulares en días seguidos.

- **Capacidad de salas:** Se refieren a la capacidad física de los cursos donde se impartirán las clases regulares (incluyendo laboratorios) o donde se tomarán exámenes.

- **Continuidad:** Son cualquier grupo de restricciones que aseguren características de una *"tabla de tiempo"* (horario de clases) de estudiantes, por ejemplo, un mismo curso (grupo de estudiantes que toman las mismas materias o el mismo examen) debería etar programado en una misma aula.

1.2. Enfoques de soluciones para Calendarización

1.2.1. Definición de Heurística y Metaheurística

Los problemas de *Calendarización Educacional* son aplicados a centros educativos de todos los niveles: Escuelas, Colegios, Universidades. Este trabajo trata de resolver calendarizaciones para *Universidades* las cuales tienen características propias incluso diferentes entre instituciones del mismo tipo, por ejemplo: Porcentaje de profesores a tiempo completo, aulas disponibles, número de laboratorios de computación, etc.

El esfuerzo (computacional y humano) de quien o quienes planifican los horarios de clases en cualquier universidad ha hecho que el problema de calendarización para este caso particular sea considerado uno de los más relevantes en la optimización combinatoria. En muchos de los casos las soluciones encontradas de forma manual suelen dejar muchas de las restricciones del problema sin cumplir, y el descontento de los actores que de una u otra forma están relacionados con la construcción del horario final, se hace notar de diferentes maneras.

Un camino para resolver el problema es la **Programación Lineal Entera (ILP)** [3], que considera un conjunto de ecuaciones que representan las restricciones naturales de la elaboración de horarios, además de una función objetivo (**FO**) que puede representar como lo hizo Molina [5] *"el número de cruces de horarios en la solución"* y que debe ser minimizada con la esperanza que tome el valor de cero (0). Formulaciones de **programación lineal** del problema y sus restricciones, se pueden encontrar a lo largo de la literatura, sin embargo muchas de ellas no contienen algunas restricciones de fuerte interés y que se esperan se cumplan en la planificación. Muchas veces, esto es debido a que algunas restricciones suelen ser difíciles de formular y limitan el problemas a instancias que no son aplicables a algunos de los casos reales. Aquí es donde se consideran alternativas de solución **Heurísticas** o **Metaheurísticas** como **Algoritmos Genéticos** [19], **Búsqueda Tabú** [7], **Recocido Simulado**, entre otros, que encuentran soluciones aproximadas, pero que con un buen diseño pueden estar muy cerca del óptimo.

La palabra Heurística proviene del griego *Heuriskein* que significa *"encontrar"*, lo cual no parece ser una palabra adecuada para describir estos métodos, porque más que *"encontrar" "busca"*.

Definición 1.2.1 (Heurísticas)**.** *Algoritmos simples, a menudo basados en el sentido común, que se supone ofrecerán una buena solución (aunque no necesariamente la óptima) a problemas específicos considerados difíciles, de un modo fácil y rápido.*

La principal diferencia entre *heurísticas* y *metaheurísticas* es que las primeras se

diseñan para problemas particulares y solo para estos, y las segundas pueden utilizarse para resolver una gran variedad de problemas combinatorios adaptando su esquema general al problema particular cambiando algunas partes de su estructura.

Los *Algoritmos Genéticos*, por ejemplo, han sido útiles para resolver la *Planificación de Proyectos con Recursos Restringidos* [12], así como para problemas de *Coloración de Grafos* como se menciona en Díaz [4].

Definición 1.2.2 (Metaheurísticas)**.** *Algoritmos basados en el sentido común o en procesos de otras áreas de la ciencia o de la naturaleza, que sirven para resolver muchos problemas considerados difíciles, encontrando soluciones aproximadas al óptimo en un tiempo pequeño.*

1.2.2. Resumen de algunos métodos de solución

Referencias específicas sobre los tres métodos presentados a continuación pueden encontrarse en Díaz [4].

- **Algoritmos Genéticos (GAs):** Estos algorítmos están inspirados en la teoría de la evolución de Darwin, Simulando el proceso de selección de la naturaleza, con la esperanza de que así se consigan éxitos similares en relación con la capacidad de adaptación de un amplio número de ambientes diferentes. La información hereditaria en los organismos es pasada a través de los cromosomas que contienen la información de todos estos factores, es decir, los genes, los cuales a su vez están compuestos por un determinado número de valores. Varios organismos se agrupan formando una población, y aquellos que mejor se adaptan son aquellos que tienen mayor probabilidades de sobrevivir y reproducirse. Algunos de los sobrevivientes son seleccionados para crear nuevos organismos. Además los genes de un cromosoma pueden sufrir cambios produciendo mutaciones en los organismos.

- **Búsqueda Tabú (Tabu Search):** Es un tipo de búsqueda por entornos, el cual guía un procedimiento de búsqueda local para explorar el espacio de soluciones más allá del óptimo local. La búsqueda Tabú selecciona de una manera agresiva el mejor de los movimientos posibles a cada paso. Al contrario que en la búsqueda local que siempre se mueve al mejor de su entorno y finaliza con la llegada a un óptimo local, la búsqueda Tabú permite moverse a una solución de su entorno o vecindad que no sea tan buena como la actual, de tal manera que pueda tener oportunidad de salir de óptimos locales y continuar estratégicamente la búsqueda de soluciones aún mejores.
  Para evitar ciclos clasifica un determinado número de los más recientes movimientos como *"movimientos Tabú"*. Por lo tanto el escape de los óptimos locales se produce de forma sistemática y no aleatoria. En otras palabras la Búsqueda Tabú mantiene una memoria de eventos. La filosofía de esta técnica es la creencia de que la elección de una mala estrategia sistemática de búsqueda es mejor que una buena elegida al azar.

- **Recocido Simulado (Simulated Annealing):** Es un método heurístico que guarda relación con un campo muy diferente al de la optimización, la termodinámica. En cada iteración una vecindad es generada (Un *horario factible* se modifica ligeramente de forma aleatoria para crear uno nuevo también factible). Este vecino es aceptado como el actual horario si se considera que tiene baja penalidad. Si por otra parte, este mismo vecino tiene una alta penalización, esta podría ser aceptada como la actual solución, es decir como un calendario (horario) acorde a una probabilidad relacionada con un parámetro de control denominado temperatura. La temperatura, y por lo tanto la probabilidad de que vecinos con alta penalidad sean aceptados, va disminuyendo en cada iteración o más usualemente después de un número de iteraciones (este número puede ser constante o puede incrementarse de acuerdo a como disminuye la temperatura).
  La relación de este método con la termodinámica está en el hecho de que el proceso se asemeja al enfriamiento en el recocido de metales.

Otras metodologías de solución son muy novedosas, como el *El Algoritmo de Búsqueda Dispersa* (*SSA* por sus siglas en inglés *Scatter Search Algorithm*) [13], **GRASP** (*Greedy Randomized Adaptive Search Procedure*) y **Redes Neuronales** que suelen considerar restricciones poco comunes en relación a los modelos estándares.

1.3. Descripción del CB-CTT

1.3.1. Entidades que participan en el CB-CTT

Como se menciona Di Gaspero [1], al momento de hacer una revisión de la literatura del **CB-CTT** se pueden encontrar una serie de documentos que presentan nuevas formulaciones pero que no siempre consideran las previamente definidas por otros autores, lo cual es debido a que no es posible escribir una que contenga todos los casos posibles que pueden presentarse en la vida real, cada institución educativa posee características diferentes que hacen que el problema se vuelva muy particular.

Con el fin de tener una descripción general del **CB-CTTP** se ha tomado como referencia la propuesta por Di Gaspero [17] que contiene las restricciones más populares (duras y suaves) del problema.

Primero se comienza con definir las entidades que participan en el problema:

- **Días (days), ranuras de tiempo (timeslots) y periodos (periods):** Se tiene un número de *días* a la semana disponible para enseñar (*teaching days*), los cuales se reparten en un número fijo de *ranuras de tiempo* iguales en todos los días. Un *periodo* es un par ordenado de la forma (*día*,*ranura de tiempo*). El número total de *periodos* a programar son el producto de días por *ranuras de tiempo*.

- **Materias (courses) y profesores (teachers):** Cada materia consiste en un número fijo de *lecturas* (*una hora clase de una materia*, viene del inglés *lecture*,

que aunque se traduce como *conferencia* simplemente se dirá *lectura*) que se programarán en distintos *periodos*, donde asisten un número determinado de *estudiantes* y es impartido por un *profesor*. Cada *materia* debería impartirse en un mínimo número de *días* a la semana, considerando además que existen periodos en los cuales las *materias* no pueden ser programadas por alguna razón definida.

- **Aulas (rooms):** Cada *aula* tiene una capacidad definida como el *número de lugares disponibles para los estudiantes* (por ejemplo pupitres).
- **Currículum (curricula):** Se define como un conjunto de *materias* tal que cualquier par de ellas en el grupo tienen *estudiantes* en común.

1.3.2. Restricciones del CB-CTT

Luego de cada restricción que se menciona a continuación, se define cuando ocurre una *violación* a ésta, con el propósito de saber que tan lejos se está de una solución *perfectamente factible*.

Restricciones Duras (HARD constraints)

- **Sesiones de clases (lectures):** Todas las *sesiones de clases* de las *materias* deben programarse y asignarse a diferentes *periodos*. Si la *sesión de clase* no está programada, se considera una *violación*.
- **Aulas Ocupadas (room occupancy):** Dos *sesiones de clases* no pueden ser programas en la misma *aula* en el mismo *periodo*. Se cuenta como una *violación* adicional cualquier *sesión de clase* extra en una misma *aula* y *periodo*.
- **Conflictos (conflicts):** *Sesiones de clases* de las materias del mismo *curriculum* o dictadas por un mismo *profesor* deben programarse en diferentes *periodos*. Una *violación* ocurre si dos *sesiones de clases* están en *conflicto*.
- **Disponibilidad (availabilities):** Si un *profesor* de una *materia* no está disponible para dictarla en un *periodo* dado, entonces ninguna *sesión de clase* de esta *materia* debe ubicarse en ese *periodo*. Cada *sesión de clase* en un *periodo* no disponible para esa *materia* se cuenta como una *violación*.

Restricciones Suaves (SOFT constraints)

- **Capacidad de Aulas (Room Capacity):** Para cada *sesión de clase* el número de *estudiantes* que asisten a las *materias* debe ser menor o igual al número de *lugares disponibles* de todas las *aulas* que acogen a las *sesiones de clases*. Se cuenta como una *violación* a cada *alumno* por encima de la capacidad del *aula*.
- **Días de Trabajo Mínimo (Minimum Working Days):** Las *sesiones de clases* de cada *materia* deben ser dictadas en un número mínimo de *días*. Cada *día* por debajo del mínimo se consideran 5 puntos de *penalidad*.

- **Curriculum Compacto (Curriculum Compactnees):** *Sesiones de clases* que pertenecen a un mismo *curriculum* deberán estar juntas unas con otras (periodos consecutivos). Para un *curriculum* dado se considera una *violación* cada vez que una *sesión de clase* no sea adyacente a otra en el mismo día.

- **Estabilización de Aulas (Room Stability):** Todas las *sesiones de clases* de una *materia* deben ser dictadas en una sola *aula*.

En [1] se consideran a las tres últimas restricicones suaves (*Días de Trabajo Mínimo*, *Curriculum Compacto* y *Estabilización de Aulas*) como *componentes opcionales de costos*, incluyendo además:

- **Carga Mínima y Máxima (Student Min Max Load)**: Para cada *currículum*, el número de *sesiones de clases* diarias deben estar dentro de un rango dado. Se considera una *violación* a cada *sesión de clase* por encima o por debajo del rango impuesto.

- **Distancia de Traslado (Travel Distance):** Todos los *estudiantes* deben tener el tiempo suficiente para trasladarse de un aula a otra, las cuales podrían estar incluso en diferentes edificios, entre dos *sesiones de clases*. En esta restricción se cuenta como una *violación* cada vez que se presente un movimiento instantáneo en dos *sesiones de clases* en *aulas* localizadas en diferentes edificios en dos *periodos* adyacentes en un mismo *día*.

- **Idoneidad de Aulas (Room Suitability):** Cuando una *materia* requiere equipos especiales (computadoras, proyector, etc.) en un *aula* específica, se dice que ésta no es idónea. Cada *sesión de clase* de una *materia* programada en un *aula* no idónea se considera como una *violación*.

- **Sesiones de clases dobles (Double Lectures):** Cuando se requiere que las *sesiones de clases* de una *materia* programadas en un mismo *día* estén en *ranuras de tiempo* adyacentes y en la misma *aula*. Cada *sesión de clase* no agrupada se cuenta como una *violación* a la restricción.

1.3.3. Un modelo de Programación Lineal

Primero es necesario establecer los parámetros y variables que intervienen en el modelo lineal, para luego definir las restriciones.

Parámetros

m = Número de *aulas*
n = Número *materias*
p = Número de *profesores*
q = Número de *periodos*
$[a, b]$ = Intervalo de *materias* correspondiente a un *currículum*
$[c, d]$ = Intervalo de *periodos* correspondiente a un *día* de la semana
K_i = Capacidad del *aula* i. $\forall i \in \{1, \ldots, m\}$
C_j = Número de *estudiantes* en la *materia* j. $\forall j \in \{1, \ldots, n\}$

Variables

$$X_{ijkl} = \begin{cases} 1 & \text{Si el } \textit{aula } i \text{ es asignada a la } \textit{materia } j \text{ con el } \textit{profesor } k \text{ en el } \textit{periodo } l \\ 0 & \text{Si no} \end{cases}$$

$\forall i \in \{1, \ldots, m\}$, $\forall j \in \{1, \ldots, n\}$, $\forall k \in \{1, \ldots, p\}$ y $\forall l \in \{1, \ldots, q\}$

Restricciones

1. Toda sala i debe ser asignada a un solo periodo.

$$\sum_{j=1}^{n} \sum_{k=1}^{p} X_{ijkl} \leq 1 \qquad \forall i, l$$

2. Toda *materia* j tiene P_j *periodos* semanales.

$$\sum_{i=1}^{m} \sum_{k=1}^{p} \sum_{l=1}^{q} X_{ijkl} = P_j \qquad \forall j$$

3. Toda *materia* j debe ser dictada por un solo *profesor* k, en una sola *aula* i y en un solo *periodo* l.

$$\sum_{i=1}^{m} \sum_{k=1}^{p} X_{ijkl} \leq 1 \qquad \forall j, l$$

4. Todo *profesor* k debe dictar no más de una *materia* j en una única *aula* i y en un solo *periodo* l.

$$\sum_{i=1}^{m} \sum_{j=1}^{n} X_{ijkl} \leq 1 \qquad \forall k, l$$

5. No deben programarse *materias* de un mismo *curriculum* en un mismo *periodo*.

$$\sum_{k=1}^{p} \sum_{j=a}^{b} \sum_{i=1}^{m} X_{ijkl} \leq 1 \qquad \forall l$$

6. La cantidad de *estudiantes* de la *materia* j debe ser menor o igual a la capacidad de la sala i.

$$X_{ijkl} \leq 1 + (K_i - C_j) \qquad \forall i, j, k, l$$

La función objetivo, puede ser formulada de más de una forma, un ejemplo puede encontrarse en Molina [5], por este motivo se ha dejado su análisis para la siguiente sección. Hay que notar, como ya se ha discutido, que es muy complicado formular algunas restriciones, y como se puede ver, este modelo puede mejorarse al incluirlas, sobre todo algunas de tipo *suaves*.

Capítulo 2

El Algoritmo Genético para el CB-CTT

2.1. Introducción a los Algoritmos Genéticos

En el presente trabajo se ha optado por un método de solución *Metaheurístico* basado en la evolución de las soluciones (factibles o no) encontradas en diferentes iteraciones de un algoritmo al que se denomina *Genético*, en parte por la experiencia de trabajo con este tipo de modelos a través de la preparación de posgrado y en parte por la gran cantidad de bibliografía disponible sobre el tema, así como el hecho de que es la base de otros métodos que actualmente están en pleno desarrollo y cuya profundización por medio de la investigación en aplicaciones a problemas complejos, pueden dar nuevas luces para otros procedimientos del mismo tipo.

A continuación se dará una descripción de la forma general de los *Algoritmos Genéticos* y luego se dará un enfoque particular al problema de *Calendarización por Materias Basado en Plan de Estudios* **CB-CTT**.

2.1.1. Relación con la evolución

John Holland en su libro *Adaptation in Natural and Aritificial Systems* del año 1975 [8] fue el primero en utilizar el término *Algortimo Genético* (al que abreviaremos **AG**), y que como su nombre lo indica, son procedimientos sistemáticos basados en la seleción natural de los seres vivos y el paso de información genética generación a generación. Holland basó su trabajo en la *evolución* de las especies, propuesta por Darwin. Esta contribución de Holland, ha servido de inspiración para crear un campo de investigación que se ha desarrollado mucho más allá del trabajo original de los *AG*.

Definición 2.1.1 (Algoritmo Genético [4])**.** *Un algoritmo Genético es una estructura de control que organiza o dirige un conjunto de transformaciones y operaciones diseñadas para simular los procesos de evolución.*

Los primeros algoritmos de Holland eran simples, pero populares gracias a que podían resolver poblemas que al menos en esa época eran considerados difíciles.

Los AG han evolucionado gracias a la contribución de muchos autores a partir de ese entonces, incluso el mismo Holland ha incorporado procedimientos más complejos y los ha mostrado en publicaciones en los años noventas. Algunas modificaciones han resultado en variantes *híbridas* que mezclan conceptos de otras heurísticas y metaheurísticas conocidas.

Lógicamente los términos utilizados por los AG en sus implementaciones computacionales guardan relación con los de la *evolución natural* y por este motivo es necesario describir algunas características de este proceso, más aún si lo que se requiere es un *proceso de evolución simulado*, por medio de un algoritmo matemático implementado computacionalmente [4] (pág 70).

Todas los cambios ocurridos a lo largo del cambio generacional ocurren en una unidad llamada *cromosoma*, que identifica a cada miembro de una *población* de individuos, de los cuales nos interesa la probabilidad que sus características genéticas sobrevivan en el futuro. La idea es que al transcurrir el tiempo en la población solo queden los más fuertes, aquellos que producen *buenas soluciones* al problema que se plantea. Los *Genes* es la estructura más simple que forma un *cromosoma*, cada miembro de una *población* tiene el mismo número de *genes*.

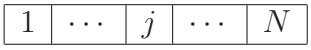


Figura 2.1: Representación de un cromosoma. N representa el número total de genes.

Los primeros algoritmos **AG** representaban a los *genes* por medio de valores binarios $1 - 0$, sin embargo esto no es aplicable en la mayoría de los problemas prácticos.

La principal característica del proceso evolutivo es la supervivencia, la elección natural del más fuerte, lo que análogamente quiere decir en un problema de optimización un individuo que procure un mejor valor de una función objetivo (FO) a minimizar (o máximizar, de acuerdo al contexto de trabajo), es decir un individuo puede ser visto como una solución posible que puede mejorar o empeorar a FO. Esto lleva a considerar una función que permita medir que tan bueno o malo es un individuo respecto a otro. El *cruce* de los miembros de una *población*, producen *hijos* con características que heredan de sus padres, donde cada pareja se elige principalmente por su fortaleza dentro del grupo al que pertenecen. La forma de *cruzar* genes de un cromosoma es un tema de discusión amplio, debido a que cada problema puede interpretarse y representarse de más de una manera.

Otro situación que se vive en la naturaleza, es que de generación en generación siempre se producen cambios estructurales los cuales se esperan que sean para bien, sin embargo con alguna *probabilidad* un descendiente pudiera traer deformaciones genéticas que lo hagan o bien vulnerable al medio ambiente o bien lo conviertan en una entidad superior, a esto se lo suele llamar *mutación*.

Muchos autores han implementado **AG** para resolver problemas combinatorios de

extrema dificultad, como lo hicieron Milena Karova, Vassil Smarkov y Stoyan Penev [9], donde desarrollan procedimientos de *cruce* y *mutación* para el conocido problema del *Traveling Salesman Problem* **TSP** y Maroto junto a Javier Alcaraz en [12], para un problema de *Planificación de Proyectos con Recursos Restringidos* que se mencionó en el capítulo anterior.

2.1.2. Definiciones Formales en los AG's

Es importante formalizar algunas definiciones.

Problema de optimización combinatoria

Se supondrá que se tiene un *espacio de búsqueda* de soluciones discreto al que llamaremos χ, y una función $f : \chi \mapsto \mathbb{R}$, el problema radica en buscar soluciones en χ para minimar f, es decir: $\text{mín}_{\mathbf{x}\in\chi} f$ (o máx sabiendo que $\text{mín}\, f = -\,\text{máx}\,(-f)$), donde $\mathbf{x}$ es un vector de *variables de decisión* y f es la *función objetivo*. A esta estructura se la conoce con el nombre de *problema de optimización combinatoria o discreta*.

El vector $\mathbf{x}$ es representado por una cadena s de largo l constituído de símbolos de un alfabeto $\mathcal{A}^l$ gracias al mapping $c : \mathcal{A}^l \mapsto \chi$. Como es posible que algunos elementos en el conjunto de llegada de c no sean soluciones factibles, podemos pensar que en realidad se necesita un subconjunto de $\mathcal{A}^l$ al que llamaremos S, entonces $S \subseteq \mathcal{A}^l$. Los elementos del string corresponden a los *genes*. A esto se lo llama *mapping genotipo-fenotipo*.

Definición 2.1.2 (Genotipo, Colin Reeves [6])**.** *Representación codificada de las variables.*

Definición 2.1.3 (Fenotipo, Colin Reeves [6])**.** *Conjunto de variables en sí mismas.*

El problema de optimización combinatoria puede escribirse entonces como:

$$\min_{s\in S} g(s)$$

donde $g(s) = f(c(s))$.

Representación de variables, cruce y mutación

Como se mencionó en la sección anterior, la representación de variables en los **AG's** fue en principio utilizando cadenas s binarias, es decir $\mathcal{A} = \{0, 1\}$. En los **AG's** se utiliza una *Población* de cadenas las cuales se denominan *cromosomas* en el mismo contexto de las sección 2.1.1. La recombinación de cadenas es lo que hemos llamado *cruce*. Un ejemplo muy simple de *cruce* con representación binaria es cuando se realiza generando números aleatorios (2 por ejemplo) que indican desde donde un *hijo* o *hija* mantienen los genes del *padre* (o de la *madre*). En el ejemplo siguiente, se fijan los puntos 4 y 8 de *cromosomas* de tamaño 10, en el primer caso, el *hijo* toma los *genes* del *padre* desde el *gen* 1 hasta el *gen* 4 y desde el *gen* 8 al 10, los

genes intermedios corresponden a los de la *madre*, el caso contrario ocurre con la *hija*, ver Figura 2.2.

			1				**2**			Números Aleatorios
1	0	1	**0**	0	0	1	**0**	0	1	Padre
0	1	0	**0**	0	1	1	**1**	0	1	Madre

1	0	1	**0**	0	1	1	**0**	0	1	Hijo
0	1	0	**0**	0	0	1	**1**	0	1	Hija

Figura 2.2: Ejemplo de cruce de dos puntos con genes binarios

Un cambio en los *genes* se denomina *mutación*. En el caso binario una mutación es cambiar de *gen* de valor 1 a valor 0 y viceversa, ver Figura 2.3.

1	0	1	0	0	**0**	1	0	0	1	Cromosoma original
1	0	1	0	0	**1**	1	0	0	1	Cromosoma mutado

Figura 2.3: Mutación en una representación binaria

La busqueda de la solución se consigue mediante la evaluación de la *función objetivo* f para cada cadena s de la *población*. A la función de evaluación se la conoce con el nombre de *fitness* y el procedimiento requiere que la cadena s con mayor valor de *fitness* pueda ser identificado, para asignarle mayor probabilidad de reproducirse.

2.1.3. Procedimientos Principales en los AG's

Población Inicial

Antes de comenzar a buscar soluciones a un problema particular, es necesario contar una un conjunto de soluciones (*Población Inicial*) el cual pueda evolucionar generación tras generación. La *Población Inicial* puede ser construida de forma aleatoria o mediante algún procedimiento sistemático. Es muy importante considerar que este conjunto debe contener suficiente información para que su transformación procure una solución final óptima o al menos muy cerca del óptimo, una *población* muy homogénea podría tener como consecuencia una convergecia hacia una solución no necesariamente buena. Debería además tener un tamaño adecuado para evitar un bajo rendimiento del algoritmo, por otro lado una tamaño de la *población* muy grande podría llevarnos a un tiempo de proceso fuera de los límites de lo esperado.

Selección para una Nueva Generación

Cuando ya se tiene una *población inicial* diversa y de calidad se debe escoger los candidatos a quienes se aplicarán los procedimientos de *cruce* y/o *mutación*, es claro que se requiere que estas soluciones deben ser individuos fuertes, lo que simula la elección natural de las especies. Existen varios métodos para seleccionar soluciones

entre los que se encuentran dos de las más conocidas: *Torneo* y *Rueda de la Fortuna* (*Roulette Wheel*).

En el método de selección por *Torneo* se eligen aleatoriamente desde la *población* dos individuos (soluciones) y de estos dos se selecciona aquel que al evaluarlo en la *función objetivo* obtenga el mejor resultado deseado (maximiza o minimiza). Existe otras variantes no se expondrán en este documento, sin embargo una muy buena referencia se encuentra en [6] pág 67.

Para el caso de la *Rueda de la Fortuna* los elementos de la *nueva generación* se obtienen por medio de la generación de una variable aleatoria discreta que se define de la siguiente forma:

1. Se ordena la *Población Actual* de mayor a menor de acuerdo a la valoración de cada cromosoma en relación a la *función objetivo*.

2. Se suman las evaluaciones de la *función objetiva* de todos los *cromosomas*.

3. A cada *cromosoma* se le asigna una probabilidad de ser seleccionada de forma aleatoria, con $Evaluacion = \frac{Evaluacion(i)}{\sum Evaluacion(Poblacion)}$.

Se selecciona un *padre* y una *madre* según el peso dado por la *función de evaluación* y la tabla de frecuencias relativas que se construye. Un ejemplo se muestra en la Figura 2.4, que se ha tomado desde Sóley [18], donde se tiene 5 individuos en la *población*, el individuo $P1$ tiene un valor de *fitness* de $f1$, $P2$ de $f2$, etc. y donde se observa que si la ruleta diera vuelta se dentendría con más probabilidad en $P3$ y que con menos frecuencia se obtendrá $P4$.

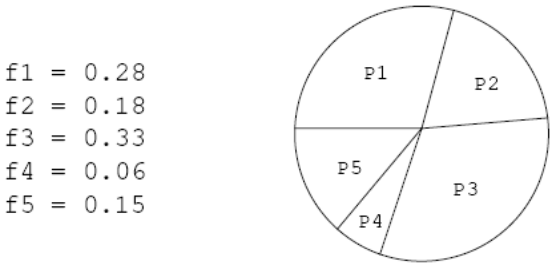


Figura 2.4: Ejemplo del método de selección *Ruleta de la Fortuna.*

Criterio de parada

Además de los procesos de *cruce* y *mutación* que ya se han explicado, es necesario contar con algún criterio de parada del algoritmo. Es claro, que mientras avanza el tiempo de ejecución del **AG** las nuevas generaciones serán cada vez más homogéneas, por este motivo un criterio natural de parada es detener el procedimiento cuando un gran número de *cromosomas* de la *población* sean iguales. Otro criterio podría ser definir un tiempo de ejecución fijo y luego de la última iteración elegir aquella

solución representada por un *cromosoma* con mejor *función de evaluación* (*fitness*).

Un seudocódigo clásico y simple del **AG** se muestra a continuación:

```
Procedimiento Algoritmo Genético
     Generar población inicial
     Evaluar población
While NO Criterio de Parada
     Selección_población
     Cruzar población
     Mutar población
     Evaluar población
end While
```

2.2. Algoritmo Genético para Calendarización

2.2.1. Elementos fundamentales en la Calendarización

Estructura del Cromosoma

Para el problema de *calendarización*, en particular para el **CB-CTT** cada cromosoma, representa un *calendario* completo. Como ya se ha discutido, el *cromosoma* está constituido por una cadena de *genes* y en este caso cada *gen* es una estructura compuesta al menos de cinco componentes:

1. Día
2. Ranura de tiempo
3. Profesor
4. Clase (de alguna Materia)
5. Aula

Recordemos que el par (*Día*, *Ranura de tiempo*) forman un *periodo*. Un ejemplo se ilustra en la Figura 2.5. Estos 5 componentes se definieron en la Sección 1.3.1.

Muchos autores optan por diferentes representaciones, por ejemplo, en [16] se representa un individuo por medio de una matriz de números enteros, donde cada fila representa respectivamente un vector de *profesores*, *periodos* y *cursos*. Estos tres vectores son definidos por todas las posibles combinaciones de estas entidades. Si se tienen dos *profesores*, tres *periodos* y tres *materias* y si se supone que el *profesor* 0 puede impartir las materias 0 y 1, el *profesor* 1 las *materias* 0 y 2, de la misma forma el *profesor* 0 está disponible en los *periodos* 0 y 1, mientras que el 1 está disponible en los tres *periodos*, por otro lado si las *materias* 0, 1 y 2 pueden ser programados en los tres *periodos*, la matriz con las tres filas vectores se verían como:

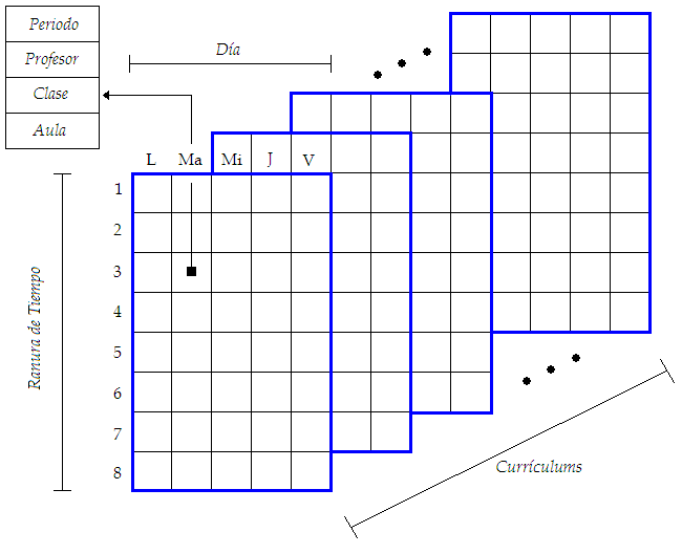


Figura 2.5: Representación de un *cromosoma* para el problema de calendarización.

$$\begin{array}{ll} \text{Vector de Profesores:} & (0,0|0,0|1,1|1,1|1,1) \\ \text{Vector de Periodos:} & (0,0|1,1|0,0|1,1|2,2) \\ \text{Vector de Materias:} & (0,1|0,1|0,2|0,2|0,2) \end{array}$$

De esta forma el *profesor* 0 puede dar clases en el periodo 0 y 1, las *materias* 0 y 1, y cada bloque representará entonces el horario disponible para cada profesor.

Estructura de la Población Inicial

La *Población Inicial* está formada por un conjunto de *calendarios*, generados de forma aleatoria, tratando de obtener diversidad y calidad en las posibles soluciones, ver Figura 2.6.

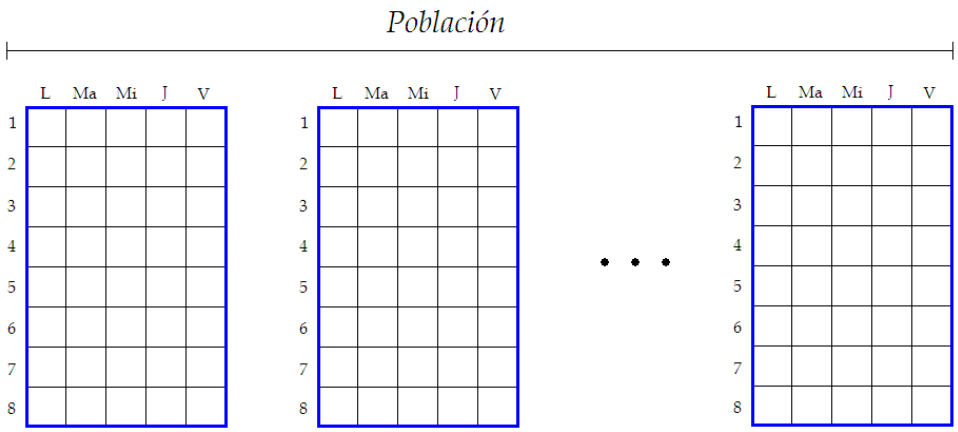


Figura 2.6: Ejemplo de *Población Inicial* para *Calendarización*.

Función de evaluación

Para este trabajo se considera una función que cuente las penalizaciones en las que se incurre si se *viola* alguna restricción, tanto *suaves* como *duras*, basado en las medidas que se mencionan en la Sección 1.3.2. Un resumen se da en la Tabla 2.1. En otras palabras esta *función de evaluación* (*fitness*) de un cromosoma determina que tan mala es la solución, y si esta medida es alta, tendrá menos probabilidad de que su información *genética* pase a generaciones futuras.

Tabla 2.1: Resumen de penalizaciones por incumplimiento de restricciones.

Restricciones	Caso	Penalización (puntos)
Restricciones Duras		
Lecturas	Lectura no programada	1
Aulas ocupadas	Lectura extra en una misma aula y periodo	1
Conflictos	Lecturas de la misma materia, currículum o dictada por un mismo profesor en un mismo periodo	1
Disponiblidad	Lectura en un periodo no disponible por un profesor	1
Restricciones Suaves		
Capacidad de aulas	Un alumno adicional en un aula con capacidad limitada	1 por cada estudiante adicional
Días de trabajo mínimo	Lecturas dictadas en un número de días no permitido	5 por cada día por debajo del mínimo
Curriculum compacto	Lecturas en un mismo día no adyacentes	1
Estabilización de aulas	Diferente aula para lecturas de una misma materia	1

2.2.2. Operadores genéticos en la Calendarización

Operador de Cruce

El *cruce* entre *calendarios padres* debe tratar de que sus descendientes posean la información genética más fuerte y que su *función de evaluación*, tal como se definió en 2.2.1, disminuya lo más posible. Para esto, este procedimiento debe evitar, en la medida de lo posible, que en sus *hijos* exista colisiones en la programación, es decir, evitar ubicar *materias* de un mismo *currículum* en un mismo *periodo* por ejemplo, ver Figura 2.7.

Una de los resultados que se ha encontrado en esta investigación, es que la idea de la implementación computacional juegan un rol muy importante en este aspecto. Un

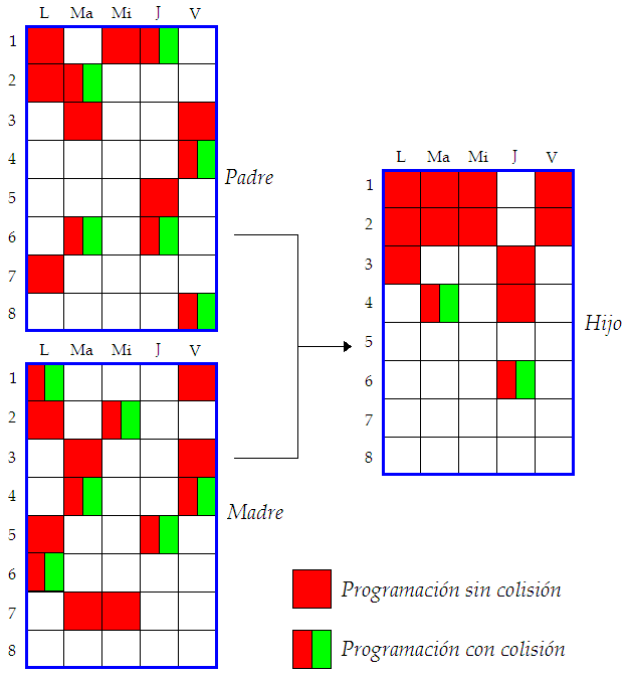


Figura 2.7: Ilustración del operador de *cruce* que evita colisiones.

algoritmo podría construir una *población inicial* con *cromosomas* donde dos *aulas* diferentes nunca se choquen en un mismo *periodo*, y quizás otra idea de programación procure *individuos* donde los *profesores* no se les asignan *lecturas* en *periodos* fuera de su disponibilidad. Incluso puede haber una generación de *cromosomas* que eviten estos dos problemas mencionados, tal como se pudo lograr en este trabajo y como se verá en el siguiente capítulo.

Operador de Mutación

El operador de mutación simplemente sigue la idea de cambiar un *gen* por otro mejor, como se ilustra en la Figura 2.8. El *gen* que mutará es seleccionado de forma aleatoria.Otra idea es incrementar el número de *genes* a cambiar por medio de generar un número aleatorio discreto menor al largo del *cromosoma*.

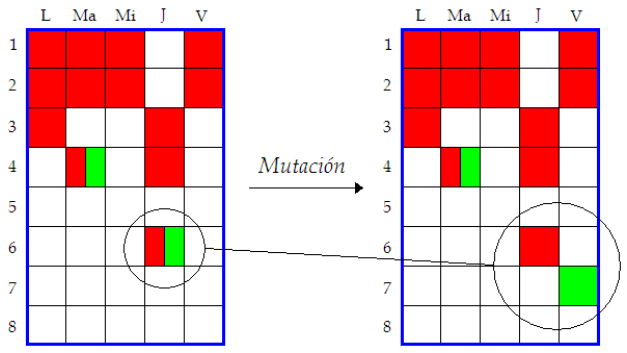


Figura 2.8: Ilustración del operador de *mutación*.

Capítulo 3

Implementación computacional de un AG para el CB-CTT

3.1. El entorno Matlab®

3.1.1. Características de Matlab®

En Rodríguez [14] se define a Matlab® [1] como un instrumento computacional simple, versátil y de gran poder para aplicaciones numéricas, simbólicas y gráficas, que contiene una gran cantidad de funciones predefinidas para aplicaciones en ciencias e ingeniería. Esta descripción de Matlab muestra muy bien las razones por la cual se ha elegido este *lenguaje de programación* para la implementación computacional de un **AG** diseñado específicamente para el **CB-CTT**.

Las funciones incorporadas en Matlab® facilitan en gran manera la programación, porque evita un gasto innecesario al momento de realizar operaciones básicas, o no tan básicas, que no forman parte del objetivo final de un algoritmo específico. Estas funciones van desde el manejo de matrices, hasta aplicaciones en Estadística e Investigación de Operaciones. Unas de las ventajas en la implementación del **AG** es el manejo de conjuntos, y el fácil manejo de estructuras de datos para búsqueda dentro de vectores de más de una dimensión, tal como la estructura del *cromosma* que se propone, ver Figura 2.5.

A lo largo de la literatura se puede observar el uso de esta herramienta en diferentes áreas de la ciencia, para el caso particular de los problemas de *calendarización para centros educativos* una referencia se encuentra en Molina [5], donde se utiliza Matlab® para la generación de horarios para la Universidad de Valparaíso en Chile, aunque la interfaz gráfica final se la realiza en utilizando el lenguaje Visual Basic® para obtener una salida de *calendarios* más sofisticada.

Algunas propiedades que caracterizan a Matlab® también se mencionan en [14]:

- Cálculo numérico rápido y con alta precisión.

[1] Matlab® es una marca registrada de The MathWorks™.

- Capacidad para manejo matemático simbólico.
- Funciones para graficación y visualización avanzada.
- Programación mediante un lenguaje de alto nivel.
- Soporte para programación estructurada y orientada a objetos.
- Facilidades básicas para diseño de interfaz gráfica.
- Extensa biblioteca de funciones.
- Paquetes especializados para algunas ramas de ciencia e ingeniería.
- Sistema de ayuda en linea.
- Iteración con otros entornos.

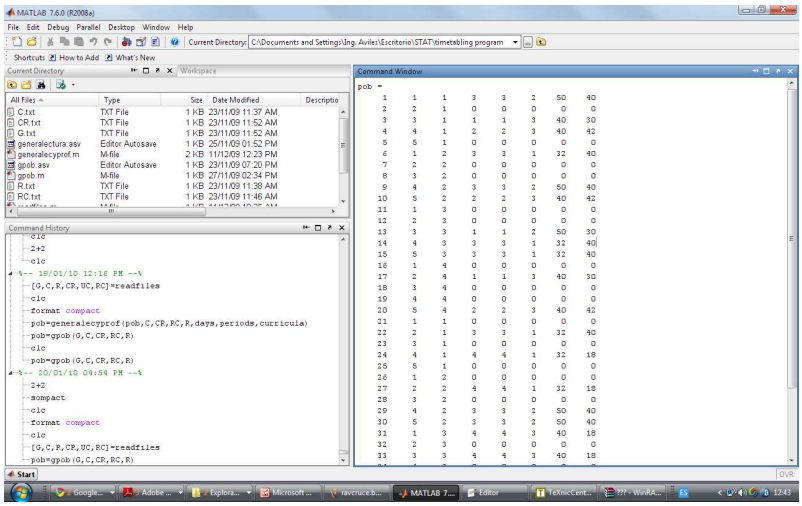


Figura 3.1: Entorno Matlab®.

3.1.2. Recursos computacionales empleados

La versión de Matlab® que se ha utilizado es **Matlab 7.6.0.324 (R2008a)**, y los recursos de hardware y software se detallan en la Tabla 3.1.

Tabla 3.1: Recursos para implementación computacional del **AG**

Sistema Operativo	Windows® XP Professional
Procesador	Intel® Core™2 Duo 1.06 GHz
Memoria RAM	2GB

3.2. Datos base para implementación

3.2.1. Formato de datos de entrada

La base de datos es de mucha importancia para esta implementación, porque el código llama a la información inicial desde archivos de formato plano *txt* para empezar el proceso de *calendarización*, que se relacionan directamente con un formato matricial definido específicamente para este propósito.

Unos datos de entrada que sirven de prueba para analizar la eficiencia de *heurísticas* y *metaheurísticas* para problemas de Calendarización (*scheduling*) y en particular para problemas de tipo **CB-CTT** se pueden encontrar en la página web del *SaTT: Scheduling and Timetabling Group*[2], aquí se define un formato de datos para inicio al que llaman **ECTT** (**Extended Curriculum-based Course TimeTabling format)**, justamente para el **CB-CTT**, que contiene tablas con información específica que guardan relación con el formato requerido para la competición mundialmente conocida **ITC** (**Internacional Timetabling Competition**) De Cesco [1], donde los participantes deben remitirse a esta forma de ingreso, porque entre otras cosas, permite hacer ejecuciones de prueba para algunos ejemplos generados para este fin y compartir experiencia entre investigadores de forma estandarizada.

Las tablas a las que hacemos referencia, contienen las siguientes variables de inicio:

Tabla: Datos generales

- Nombre para el problema a ejecutar.
- Número de aulas.
- Número de días para calendizar.
- Número de periodos por días.
- Número de currículums.
- Min-Max lecturas diarias.
- Número de restricciones de no disponibilidad.
- Número de restricciones de aulas.

Tabla: Materias

- Nombre de identificación para las materias.
- Nombre de los profesores que dictan las materias.
- Número de lecturas de cada materia.

[2]Diegm-University of Udine (Italy), main contacts: Prof. Andrea Schaerf y Dr. Luca Di Gaspero; http://tabu.diegm.uniud.it/

- Mínimo número de días para programar a las materias.
- Número de estudiantes que toman una materia.
- Valor binario 0-1 que indica si se requiere o no que dos lecturas programadas en el mismo día deban ser asignadas a periodos consecutivo (algo que ocurre en la mayoría de los casos).

Tabla: Aulas

- Nombre de identificación de las aulas.
- Capacidad de las aulas.
- Identificador que define la ubicación (en un bloque de edificio por ejemplo) de cada aula.

Tabla: Currículums

- Nombre de identificación del currículum (semestre).
- Número de cursos de los currículums.
- Nómbres de identificación de las materias en cada currículum.

Tabla: Restricciones de no disponibilidad

- Nombre de identificación para las materias.
- Días en que las materias no pueden ser programadas.
- Periodos del día (Timeslots) en que las materias no pueden ser programadas.

Tabla: Restricciones de aulas

- Nombre de identificación para las materias.
- Nombre de identificación del aula donde no pueden ser programadas las materias.

3.2.2. Formulaciones

Además de las tablas mencionadas en la sección anterior, en el **ITC** se caracterizan los problemas a resolver por medio de cinco diferentes formulaciones (ver Tabla 3.2) que contiene un específico conjunto de componentes de *costos* y los *pesos* (penalizaciones de acuerdo a la Tabla 2.1) correspondiente a cada uno de ellos, en realción a la defínción de la *función objetivo* que tan solo suma los costos por *violación* de restricciones.

En la Tabla 3.2, UD se refiere a la *Universidad de Udine* de Italia, la restricción *Currículum compacto Versión dos* se define como: *Lecturas programadas en un mismo día no deben tener lecturas de otras materias entre ellas* y "D" es la inicial de restricciones *duras*.

Tabla 3.2: Descripción de formulaciones de problemas de *calendarización*. Adaptado de De Cesco [1]. Para referencia de los nombres de los *componentes de costo* considere la Sección 1.3.2

Formulación del problema: Componente de costos	UD1	UD2	UD3	UD4	UD5
Lecturas	D	D	D	D	D
Conflictos	D	D	D	D	D
Aulas ocupadas	D	D	D	D	D
Disponibilidad	D	D	D	D	D
Capacidad de aulas	1	1	1	1	1
Días de trabajo mínimo	5	5	-	1	5
Currículum compacto	1	2	-	-	1
Currículum compacto versión dos	-	-	4	1	2
Estabilización de Aulas	-	1	-	-	-
Carga mínima y máxima	-	-	2	1	2
Distancia de traslado	-	-	-	-	2
Idoneidad de aulas	-	-	3	D	-
Lecturas dobles	-	-	-	1	-

La formulación **UD4** es de importancia para este trabajo, porque considera todas las restricciones que se espera se cumplan en la aplicación del **AG** que se propone. Adicionalmente se cumple también la restricción *Currículum Compacto* versión 1.

3.2.3. Instancias

En la página web del **SaTT** se puede encontrar una serie de problemas bajo las formulaciones dadas en la Sección 3.2.2 que son instancias que pueden bajarse de forma gratuita para hacer pruebas de algún algoritmo propuesto para rersolver el problema de *Calendarización Educacional*, 21 de estas instancias corresponden a datos reales tomados de la Universidad de Udine, codificadas con los nombres $comp01, \cdots, comp21$, ver Figura 3.2.

En las instancias consta además un pequeño ejemplo denominado **Toy** que es uno de los pocos problemas donde se encuentra un valor de la *función de evaluación* igual a cero. Este valor óptimo se lo ha realizado bajo la técnica del *Recocido simulado* que se menciona en la Sección 1.2.2.

Toy se ha utilizado para hacer las pruebas de ejecución de nuestro **AG** y sus componentes se describe en un archivo *txt* que se detalla en la Tabla 3.3 y que guardan

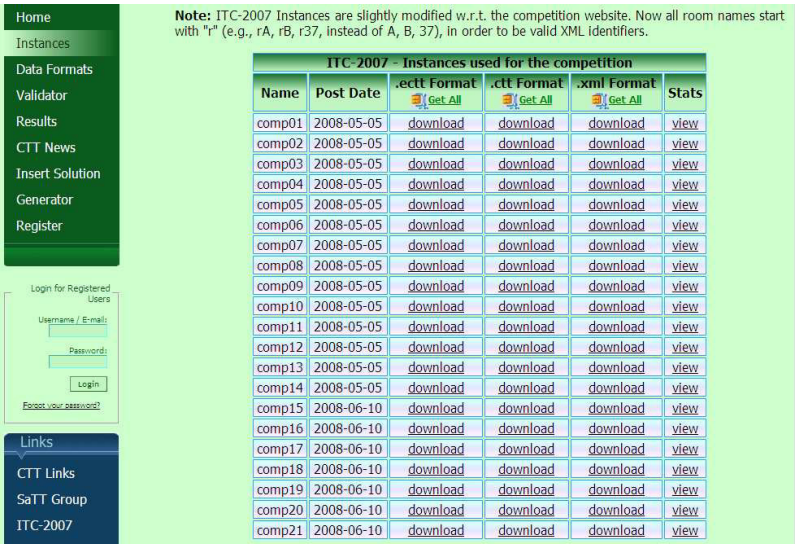


Figura 3.2: Instancias de la Universidad de Udine vista desde la página Web de **SaTT**.

relación a lo definido en el subcapítulo 3.2. **Toy** puede encontrase en cualquiera de las formulaciones descritas en la Sección 3.2.2, sin embargo para las corridas del programa se ha tomado la versión **ECTT**.

3.2.4. Matrices de entrada para el AG en Matlab®

Con el propósito de dar características fijas a los datos de entrada, se ha diseñado un esquema (*template*) para que el algoritmo que se propone pueda ejecutar las funciones y procedimientos éscritas en código Matlab®. Este esquema sigue el formato de datos de entrada descrito en el apartado 3.2.1, con el ligero cambio en los nombres de los *profesores*, código de *materias* y *aulas*, a las que se ha representado utilizando números dentro del conjunto $\mathbb{N} = \{1, 2, 3, \ldots, \infty\}$.

Se han creado archivos de extensión *txt* que serán leídos por un simple procedimiento cuyo código se puede revisar en el Programa 1. Los nombres y un código de identificación de estos archivos planos se detallan en la Tabla 3.4.

Programa 1 "readfiles.m": Procedimiento para leer archivos externos *txt*

```
G=load('G.txt');
C=load('C.txt');
R=load('R.txt');
CR=load('CR.txt');
UC=load('UC.txt');
RC=load('RC.txt');
```

Tabla 3.3: Contenido del archivo *toy.txt*

```
Name: Toy
Courses: 4
Rooms: 3
Days: 5
Periods_per_day: 4
Curricula: 2
Constraints: 8

COURSES:
SceCosC Ocra 3 3 30
ArcTec Indaco 3 2 42
TecCos Rosa 5 4 40
Geotec Scarlatti 5 4 18

ROOMS:
rA 32
rB 50
rC 40

CURRICULA:
Cur1 3 SceCosC ArcTec TecCos
Cur2 2 TecCos Geotec

UNAVAILABILITY_CONSTRAINTS:
TecCos 2 0
TecCos 2 1
TecCos 3 2
TecCos 3 3
ArcTec 4 0
ArcTec 4 1
ArcTec 4 2
ArcTec 4 3

ROOM_CONSTRAINTS:
SceCosC rA
Geotec rB
TecCos rC

END.
```

Tabla 3.4: Nombres y códigos de los archivos *txt*

Nombre de la tabla	Código para la tabla
Datos generales	G
Materias (courses)	C
Aulas (rooms)	R
Currículums (curricula)	CR
No disponibilidad (unavailability constrains)	UC
Restricciones de aulas (room constrains)	RC

Las matrices generadas por el Programa 1, se muestran en pantalla como matrices de números, los cuales son fáciles de manipular para ejecutar y apropiadas para otras funciones del **AG**, ver Figura 3.3.

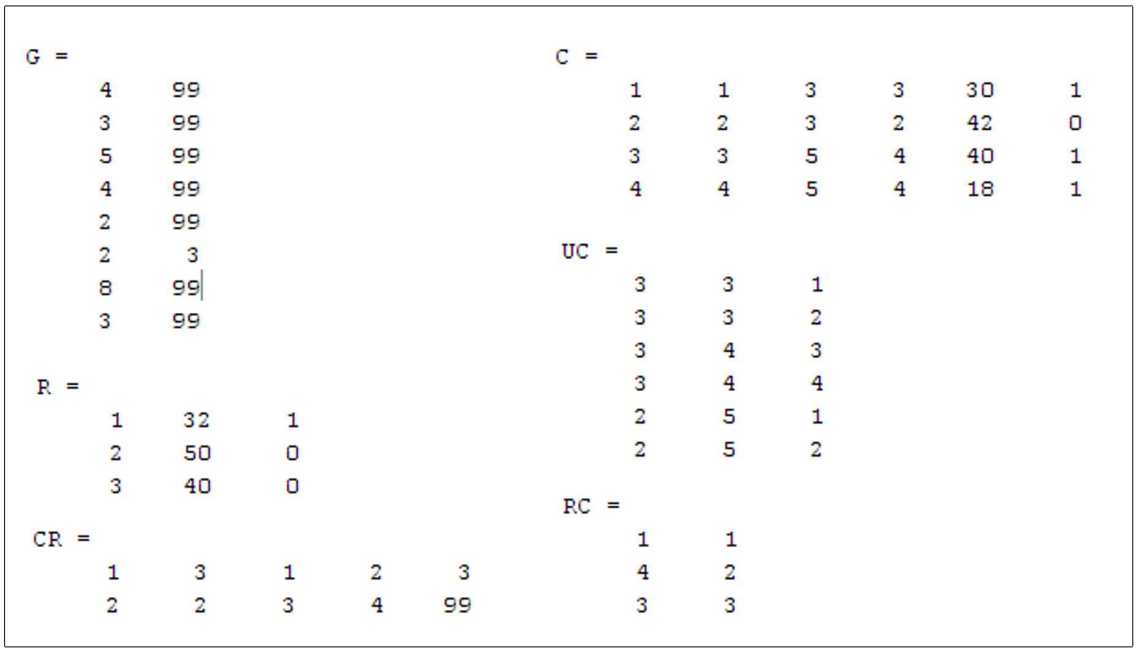


Figura 3.3: Salida de datos de entrada en interfaz Matlab®

Los números 99 representan espacios vacios dentro en las matrices formadas por los datos de los archivos de entrada *txt* que son llamados por el Programa 1.

3.3. Diseño del AG para el CB-CTT en Matlab®

3.3.1. Representación cromosómica del calendario

Un calendario se ha representado en este trabajo como una matriz de dimensión $[(días)(ranuras\ de\ tiempo\ por\ día)(currículums) \times 8]$. En este arreglo, el número de filas corresponde a cada *periodo* disponible para calendarizar multiplicado por el número de *currículums* considerados en un problema particular, esto quiere decir

que los *calendarios* de cada *currículum* se dispondrán uno debajo del otro. Por otra parte, cada columna de la matriz representa respectivamente:

1. Indicador del número de la fila 1, 2, 3,. . ., etc.
2. Día de la semana (laborables).
3. Ranura de tiempo (horas de clases disponibles).
4. *Materia* que sedictará en un periodo.
5. *Profesor* de la *materia*.
6. *Aula* a utilizar.
7. Capacidad del *aula* a utilizar.
8. Número de *estudiantes* registrados en la materia.

Índice	Día	Ranura	Materia	Profesor	Aula	Capacidad	Estudiantes
1	1	1	3	3	2	50	40
2	2	1	0	0	0	0	0
3	3	1	1	1	3	40	30
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
19	4	4	0	0	0	0	2
20	5	4	2	2	3	40	42
21	1	1	0	0	0	0	0
22	2	1	3	3	1	32	40
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
38	3	4	4	4	3	40	18
39	4	4	0	0	0	0	0
40	5	4	3	3	1	32	40

Figura 3.4: Representación matricial de un cromosoma (*calendario*)

Un ejemplo de la representación matricial de un *cromosoma* para *Toy* se observa en la Figura 3.4, donde los diferentes colores rojo y azul determinan respectivamente un calendario para el *currículum* 1 y *currículum* 2.

Una de las ventajas de esta representación matricial es que no permite que *lecturas* de una misma *materia* sean programadas en un mismo *periodo*, es decir, la restricción *dura* a la que se ha llamado *Lecturas*, ver Sección 1.3.2, se supera por completo, y toda posible solución al problema planteado tendrá cero como valor de evaluación

para esta restricción.

De esta misma forma ninguna materia de un mismo *currículum* se ubica en un mismo *periodo*, es decir, se asegura que *Conflictos*, la cual es otra restricción *dura*, se cumpla en un 50 %, porque sí podría proramarse dos materias de diferente *currículum* en una misma combinación (*día,ranura de tiempo*) pero con diferente *profesor* y en diferente *aula*.

3.3.2. Algoritmo para crear un cromosoma

La función **gcrom**, ver Programa 9, Apéndice A, se ha creado para generar la representación matricial de un *cromosoma* ubicando aleatoriamente para cada *currículum* en algún *periodo* una *materia*, un *profesor* y un *aula*, además de su capacidad y del número de estudiantes registrados, tomándolas desde los archivos *txt*'s previamente llamados por la función readfiles, ver Programa 1. La función tiene cuidado de no ubicar más de las materias definidas para cada *currículum* en el archivo *CR.txt*, ver Tabla 3.4. Todos esto se logra con una función denominada **generalecyprof** que a su ves llama a un procedimiento llamado **roomycap**, ver Programas 12 y 10 en el Apéndice A.

En este punto se aplica un procedimiento que trata de mejorar el *cromosoma* inicial cambiando aleatoriamente el *aula* antes asignada por otra siempre y cuando la *función objetivo* mejore en su valor, para este caso esto quiere decir que la restricción *suave* llamada *Capacidad de Aulas* se supere. Este procedimiento se ha nombrado como **mejoraroom**, ver Programa 11 también en el Apéndice A.

3.3.3. Programa de generación de la Población Inicial

La primera generación se construye simplemente repitiendo n veces la función **gcrom**, donde n es el tamaño de la *población*. Como ya se ha mencionado, se pretende que esta *población* contenga soluciones diversas pero también de calidad.

La función **gpob** se muestra en el Programa 2.

Programa 2 "gpob.m": Función que genera la *población inicial*

```
function poblacion=gpob(G,C,CR,RC,R,UC,n)
for i=1:n
poblacion(:,:,i)=gcrom(G,C,CR,RC,R,UC);
end
```

3.3.4. Forma de la función de evaluación

Todas las restricciones descritas en la Sección 1.3.2 se han medido considerando las penalizaciones resumidas en la Tabla 2.1. La función **fitness**, que se presenta

en el Programa 3, llama a los procedimientos **roomoccupancy**, **conflicts**, **uconstraints**, **roomcapacity**, **minways**, **ccompactness** y *rstability*, que representan a las restricciones que se consideran para el problema del **CB-CTT**, ver Tabla 3.5.

Programa 3 "fitness.m": Función de evaluación para un *cromosoma*

```
function [ffitnessv ffitness]=fitness(cromosoma,G,R,UC,C)
days=G(3,1);
periods=G(4,1);
rooms=G(2,1);
courses=G(1,1);
curricula=G(5,1);
le=days*periods*curricula;
roomoccupancy;
conflicts;
uconstraints;
roomcapacity;
minwdays;
ccompactness;
rstability;
ffitnessv=[froomoccupancy fconflicts fuconstraints froomcapacity
           fminwdays fccompactness frstability];
ffitness=sum(ffitnessv);
```

Tabla 3.5: Procedimientos de la función **fitness**

Nombre de procedimientos	Nombre de restricciones
roomoccupancy	Aulas Ocupadas
conflicts	Conflictos
uconstraints	Unavailability constraints-Disponibilidad
roomcapacity	Capacidad de aulas
minwdays	Días de trabajo mínimo
ccompactness	Currículum Compacto
ccompactnessv2	Currículum Compacto versión dos
rstability	Estabilización de aulas

Un ejemplo de la aplicación de la función de evaluación en Matlab® se presenta en la Figura 3.5. La suma del vector *fitnessv* es la medida de incumplimiento de restricciones.

3.3.5. Evaluación de una poblacion

La evaluación en la población se la realiza utilizando la función **evaluapoblacion**, ver Programa 4. Este procedimiento simplemente da el valor de *fitness* para cada

```
>> [ffitnessv ffitness]=fitness(cromosoma,G,R,UC,C)
ffitnessv =
     1     0     3    54     2     2     0     4
ffitness =
    66
```

Figura 3.5: Ejemplo de aplicación de la función **fitness**

cromosoma, presentándolos en un vector que se ha llamado **fitnessp**, ver Figura 3.6.

Programa 4 "evaluapoblacion.m": Devuelve un vector con el *fitness* de cada *cromosoma* de una *población*

```
function [fitnessp]=evaluapoblacion(poblacion,G,R,UC,C)
s=size(poblacion);
tp=s(3);
for i=1:tp
    fitnessp(i,1)=fitness(poblacion(:,:,i),G,R,UC,C);
end
```

```
>> [fitnessp]=evaluapoblacion(poblacion,G,R,UC,C)
fitnessp =
    36
    58
    76
    71
    87
```

Figura 3.6: Ejemplo del vector **fitnessp** para una población de tamaño 5

3.3.6. El proceso de selección

Para poder pasar de generación en generación se ha diseñado un procedimiento de selección de soluciones que permite elegir de entre todos los *cromosomas* de una *población* a los mejores, aquellos cuya función de evaluación sea la mejor posible. El método que se ha implementado es la *Rueda de la Fortuna* (*Roulette Wheel*) que se definió en el Apartado 2.1.3, ver Programa 5.

3.3.7. Diseño del procedimiento de cruce

En la función **cruce**, ver Programa 20, Apéndice A, se desarrolla un procedimiento sistemático, que transfiere información genética de un *padre* y una *madre* para generar un *hijo* y una *hija*.

El proceso comienza buscando en el *calendario* del *padre* y la *madre* el *periodo* donde cada una de las *materias* ya hayan sido prgoramadas. Luego se genera un número aleatorio uniforme de parámetros 1 y 0 con el fin de determinar si es el *gen* del *padre* que se tranferirá al *hijo* o el de la *madre*, y se ubicarán los datos del ascendiente en el mismo *periodo* encontrado, cuidando que esta ubicación esté libre en el descendiente, caso contrario se selecciona una ubicación libre de forma aleatoria. Cuando la elección se ha realizado, la información que se tranfiere al otro descendiente es exactamente lo opuesto, siempre que sea posible. Todo esto se lo realiza para cada *calendario* de cada *currículum*.

Un ejemplo del proceso de cruce se muestra en la Figura 3.7. El ejemplo ilustra el paso de información genética hacia los hijos utilizando los colores azul y rojo para el padre y la madre respectivamente, ubicando los *genes* en las ubicaciones correspondientes de acuerdo a lo mencionado en el párrafo anterior.

3.3.8. Función de cruce para la Población

Con el propósito de asegurar que las futuras generaciones posean miembros más fuertes, se ha decidido que al momento de cruzar dos individuos, se elija entre el

Programa 5 "seleccion.m": Selecciona individuos para nueva generación

```
function poblacion=seleccion(poblacion,fitnessp)
sp=size(poblacion);
maximo=max(fitnessp);
veclargo=[];
for i=1:sp(3);
    vecs(i)=maximo-fitnessp(i,1)+1;
    for k=length(veclargo)+1:length(veclargo)+vecs(i)
        veclargo(k)=i;
    end
end
for k=1:sp(3)
    r=ceil(length(veclargo)*rand);
    v=veclargo(r);
    nuevapoblacion(:,:,k)=poblacion(:,:,v);
end
poblacion=nuevapoblacion;
```

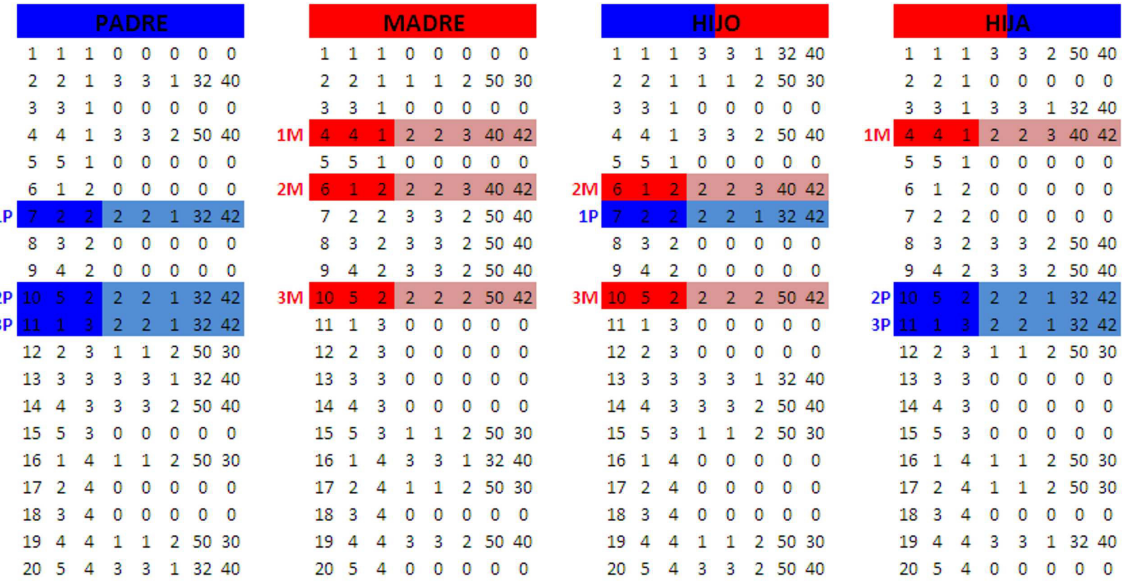


Figura 3.7: Ejemplo de un proceso de *cruce* para la base de datos **Toy**.

hijo y la hija, al de menor función de evaluación para que forme parte de la nueva *población*. La función **fitness** se aplica a ambos *cromosomas* y se verifica quien sobrevive y quien no. El padre y la madre se seleccionan de forma aleatoria desde la *población* luego del proceso de selección descrito en la Sección 3.3.6.

La función que permite este procedimiento es **crucepoblación**, ver Programa 23 en el Apéndice A. El parámetro **probcruce** representa la probabilidad de que dos individuos se crucen, y se ajusta luego de experimentos computacionales como se verá en el siguiente capítulo. En la literatura que se ha revisado este valor regularmente es alto.

3.3.9. Mutación de un cromosoma

La mutación es un procedimiento que actúa sobre los *genes*. En esta implementación se ha considerado para cada fila de la representación cromosómica no vacía, ubicar el *gen* (la fila) en un lugar disponible de forma aleatoria, pero en este caso, este cambio se produce siempre que la función de evaluación mejore en su medida. Esto asegura que si la mutación en un *gen* ocurre, esto sea para bien, ver Figura 3.8.

El valor de la función de evaluación para **Cromosoma** es 97 y para **MUTACIÓN** es 93.

La función **mutacion** produce estos cambios, ver Programa 24 en Apéndice A. En el código se puede observar, que un parámetro de entrada es **probmutacion** el cual sirve para determinar si un *gen* es candidato a mutar.

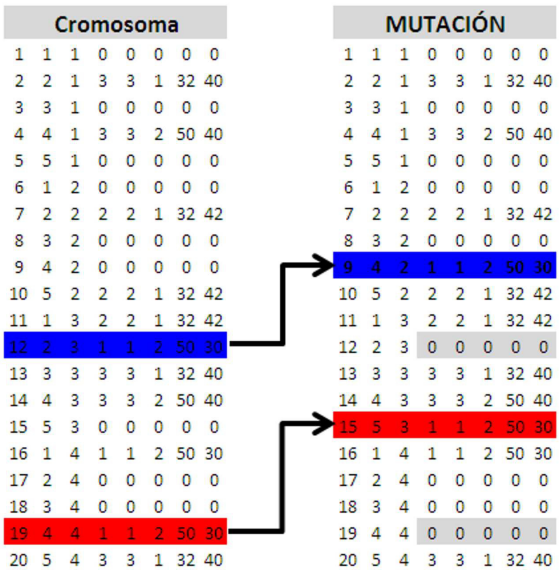


Figura 3.8: Ejemplo de un proceso de *mutacion* para la base de datos **Toy**.

3.3.10. Función de mutación para la población

El proceso de *mutación* debe aplicarse a cada elemento de la *población* de acuerdo a una probabilidad que se define en la función **mutacionpoblacion**, ver Programa 25 Apéndice A, como probmutacion, esta es la misma probabilidad dada en la función **mutacion** para decidir si un *gen* de un cromosoma mutará o no. Esto significa que la misma probabilidad servirá para definir si un *cromosoma* entra al proceso de *mutación* además de decidir si un *gen* cambiará de posición. Este parámetro debe ajustarse también luego de probar con diferentes valores en muchas corridas del **AG**.

3.3.11. Criterio de parada del AG

En teoría el **AG** podría ejecutarse en tiempo indefinido, teniendo en consideración que los elementos de la *poblacion* luego de un tiempo t grande se parecerán entre sí y se espera además que la función de evaluación tenga un valor cercano a cero (o cero en el mejor de los casos). Debido a esto un criterio de parada del **AG** se ha definido en este trabajo como:

1. Para si al menos una proporción p de individuos miembros de la *población* tienen el mismo valor de *fitness*, ó

2. Si en alguna iteración algún elemento de la *población* tiene un valor de función de evaluación igual cero

En el primer caso la solución será aquel *cromosoma* con mínimo *fitness*, si existe más de una solución encontrada, se elige una de forma aleatoria, y en el segundo caso aquel que tenga el valor de cero.
En la función **criterioparada**, ver Programa 6, se implementa el caso 1 y en el caso dos se lo considera en el programa general del **AG**, ver Programa 7.

Programa 6 "criterioparada.m": Criterio de parada caso 1, proporción de elementos con igual *fitness*

```
function cparada=criterioparada(fitnessp)
if sum(ismember(fitnessp,min(fitnessp)))>=0.8*length(fitnessp)==1
    cparada=1;
else
    cparada=0;
end
```

3.3.12. AG completo

Luego de tener todos los elementos necesarios para la implentación del **AG** solo falta enlazar todos estas rutinas en un solo programa, que permita obtener la solución deseada. A este programa general se denomina **agcbctt**, ver Programa 7. El cual guarda relación estricta con el código clásico de un **AG** dado en la Sección 2.1.3.

Programa 7 "agcbctt.m": AG completo

```
function [poblacion tt]=agcbctt(G,C,R,CR,UC,RC,n,probcruce,probmutacion)
poblacion=gpob(G,C,CR,RC,R,UC,n);
fitnessp=evaluapoblacion(poblacion,G,R,UC,C);
iteracion=0;
while (criterioparada(fitnessp) || min(fitnessp)==0) ~=1
    poblacion=seleccion(poblacion,fitnessp);
    poblacion=crucepoblacion(poblacion,G,R,UC,C,probcruce);
    poblacion=mutacionpoblacion(poblacion,G,R,UC,C,probmutacion);
    fitnessp=evaluapoblacion(poblacion,G,R,UC,C);
    iteracion=iteracion+1
end
posmin=find(fitnessp==min(fitnessp));
tt=poblacion(:,:,posmin);
```

Capítulo 4

Experimentos computacionales

4.1. Calibración de parámetros

Los valores de los parámetros de entrada del **AG** deben ser ajustados mediante un proceso de ensayo y error, es decir, probar diferentes valores hasta que el diseñador considere que ha encontrado estabilidad en la ejecución del algoritmo. Con estabilidad se quiere decir que el programa ejecutado alcanza los objetivos deseados en un tiempo razonablemente pequeño. Un resumen de los parámetros de entrada se detallan a continuación:

1. Tamaño de la *población*.
2. Probabilidad de mutación.
3. Probabilidad de cruce.

Los datos **Toy**, ver Tabla 3.3, se utilizan para la calibración, sin embargo se debe mencionar que cada problema tiene características que los hace diferentes a cualquier otro y los parámetros podría variar en su valor, por este motivo lo que se presenta a continuación es un solo un ejemplo del procedimiento para ajustar los valores requeridos.

Se ha creado un procedimiento, ver Programa 8, con el que se repite 30 veces el **AG** para diferentes valores de los parámetros a calibrar. Esto se lo realizó un número grande de veces, sin embargo se muestran en los gráficos siguientes aquellas corridas que hicieron que se tomará una decisión sobre los parámetros a utilizar para encontrar soluciones en nuestro problema de prueba.

Las Tablas 4.1, 4.2 y 4.3, contienen información de:

1. Número de corridas.
2. Probabilidad de cruce y/o mutación.
3. Tamaño de muestra.
4. Porcentaje de individuos de igual fitness para criterio de parada.

Programa 8 "pruebas.m": Genera continuas corridas del **AG**

```
x=[10 20 30 40 50 60 70 80 90 100];
suma=1;
for i=1:length(x)
    for j=1:30
        suma
        [tt iteracion t]=agcbctt(G,C,R,CR,UC,RC,x(i),0.8,0.2);
        [ffitnessv ffitness]=fitnessvector(tt,G,R,UC,C);
        ffitnessvec(i,j)=ffitness;
        iteracionvec(i,j)=iteracion;
        tvec(i,j)=t;
    end
    promfitness(i,1)=round(mean(ffitnessvec(i,:)));
    promiteracion(i,1)=round(mean(iteracionvec(i,:)));
    promt(i,1)=round(mean(tvec(i,:)));
    suma=suma+1;
end
promfitness
promiteracion
promt
```

5. Valor de *fitness* promedio en las 30 corridas para cada valor fijo.
6. Número de iteraciones promedio en las 30 para cada valor fijo.
7. Tiempo de ejecución promedio (segundos) en las 30 para cada valor fijo.
8. Promedios de valores encontrados.
9. Desviación estándar de valores encontrados.

La Tabla 4.1 muestra valores medios de las repeticiones de la función **"agcbctt"** para los valores de $n = 10, 20, \ldots, 100$. En la Figura 4.1 se puede observar que cuando se fijan los valores de probabilidad de *cruce* y *mutación* el **AG** parece encontrar soluciones óptimas. El número de iteraciones que se necesitan para que el **AG** encuentre una solución (no necesariamente la óptima) de acuerdo a la Figura 4.2 se vuelve estable, entre 22 y 26, para valores de n igual a 30, pero como se menciona antes, los *cromosomas* óptimos se empiezan a encontrar a partir de $n = 50$. La Figura 4.3 da una tendencia lineal de crecimiento mientras el tamaño de la *población* crece. Las pruebas para este ejemplo han mostrado que tiempos aceptables del algoritmo podrían estar entre 30 y 50 segundos, que en el gráfico se muestran para n entre 40 y 60.

Los valores fijos de las probabilidades de *cruce* y *mutación*, se obtuvieron luego de analizar de forma conjunta los gráficos para variaciones en estos parámetros.

Tabla 4.1: Experimento computacional (Tamaño de Población)

Parámetros	
Corridas c/pob	30
Prob. cruce	0.8
Prob. mutacion	0.4
% de igual fitness	0.8

n	Fitness	Iteraciones	Tiempo ejecución (seg)
10	10	15	4
20	3	20	11
30	1	26	21
40	1	25	27
50	0	24	33
60	0	26	45
70	0	23	45
80	0	23	51
90	0	25	62
100	0	27	74

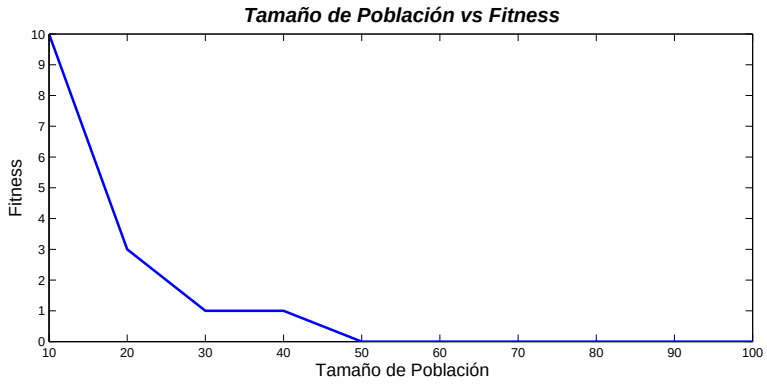


Figura 4.1: Tamaño de Población vs Fitness, variación de *población*

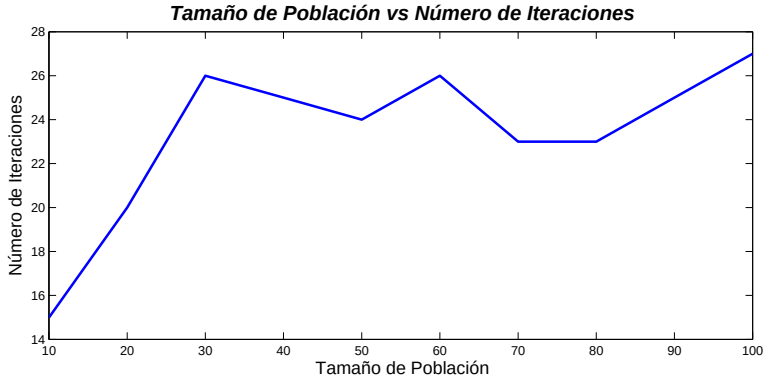


Figura 4.2: Tamaño de Población vs Número de Iteraciones, variación de *población*

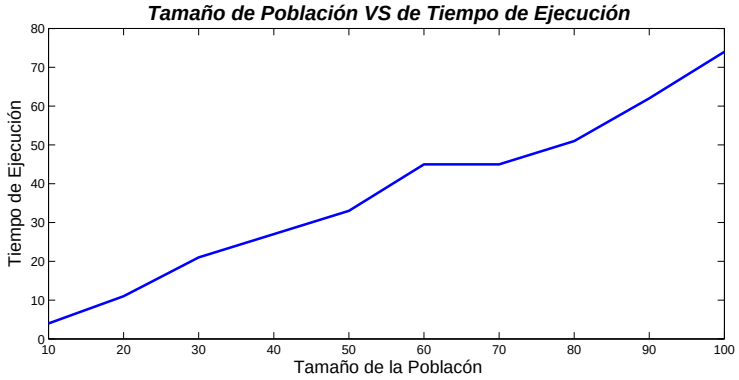


Figura 4.3: Tamaño de Población vs Tiempo de Ejecución, variación de *población*

En la Tabla 4.2 se ha hecho variar la probabilidad de cruce en valores de $probcruce = 0{,}1, 0{,}2, \ldots, 1$, y se puede observar en la Figura 4.4, que el óptimo se encuentra a partir del valor de probabilidad de cruce de 0,5, sin embargo, un análisis más detallado mostró que el valor de este parámetro podría estar entre 0,6 y 0,8. Para estas probabilidades de cruce, el promedio del número de iteraciones que se necesitaron, está entre 23 y 27, ver Figura 4.5 y su tiempo de ejecución promedio entre 40 y 50 segundos como lo presenta la Figura 4.6.

Tabla 4.2: Experimento computacional (Probabilidad de Cruce)

Parámetros	
Corridas c/pob	30
n	60
Prob. mutación	0.4
% de igual fitness	0.8

probcruce	Fitness	Iteraciones	Tiempo ejecución (seg)
0.1	8	23	16
0.2	3	26	21
0.3	1	27	27
0.4	1	26	31
0.5	0	26	36
0.6	0	27	46
0.7	0	24	45
0.8	0	23	47
0.9	0	24	56
1	0	24	62

Un análisis similar a los dos anteriores se realiza en la Tabla 4.3 y en las Figuras 4.7, 4.8 y 4.9, donde se establecen valores de probabilidad de mutación mayores de 0,5 para un número de iteraciones promedio entre 0,3 y 0,7, y un tiempo de ejecución aproximado de 50 segundos.

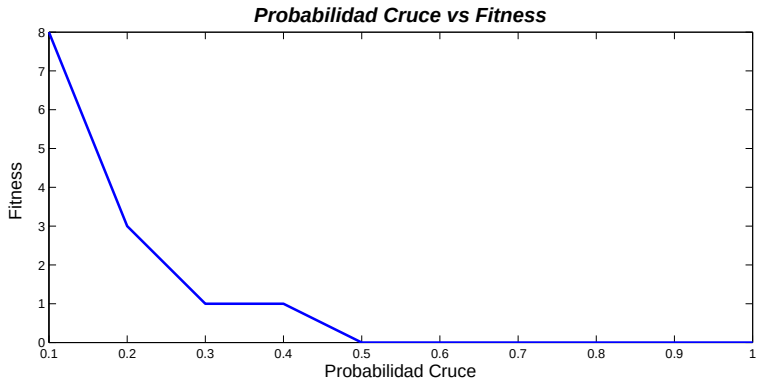


Figura 4.4: Probabilidad de Cruce vs Fitness, variación de probabilidad de *cruce*

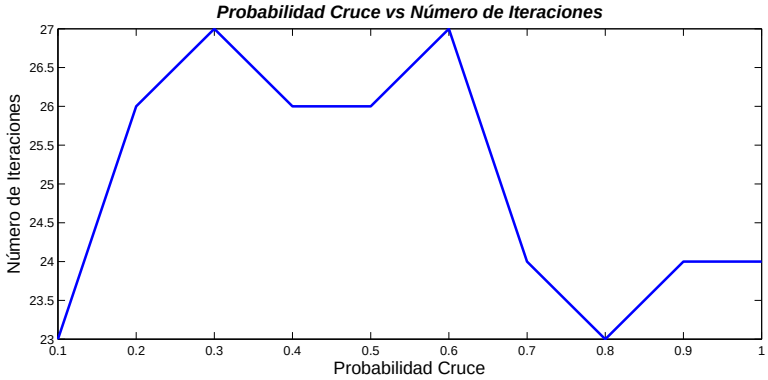


Figura 4.5: Probabilidad de Cruce vs Número de Iteraciones, variación de probabilidad de *cruce*

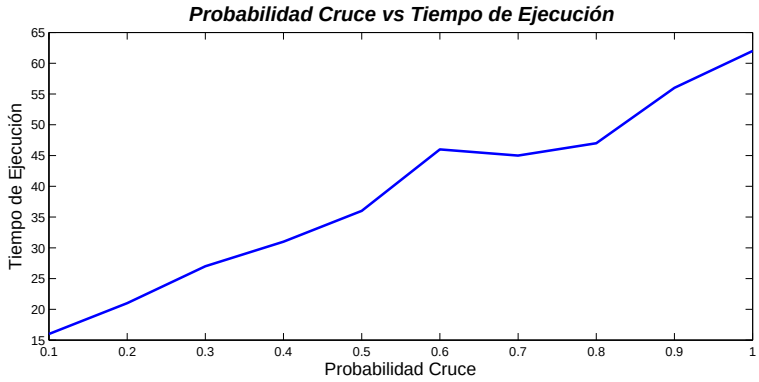


Figura 4.6: Probabilidad de Cruce vs Tiempos de Ejecución, variación de probabilidad de *cruce*

Tabla 4.3: Experimento computacional (Probabilidad de Mutación)

Parámetros	
Corridas c/pob	30
n	60
Prob. cruce	0.8
% de igual fitness	0.8

probmutacion	Fitness	Iteraciones	Tiempo ejecución (seg)
0.1	7	19	11
0.2	4	22	18
0.3	1	24	26
0.4	1	24	37
0.5	0	23	50
0.6	0	24	63
0.7	0	22	72
0.8	0	23	73
0.9	0	24	77
1	0	25	83

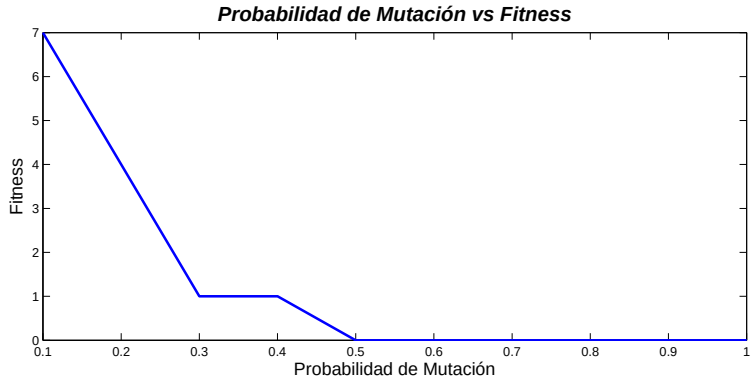


Figura 4.7: Probabilidad de Mutación vs Fitness, variación de probabilidad de *mutación*

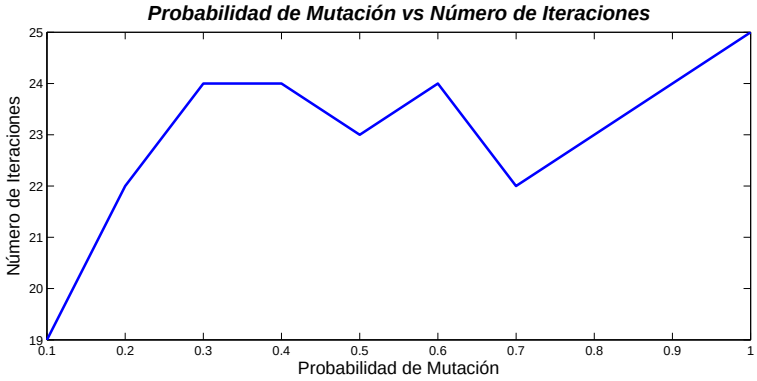


Figura 4.8: Probabilidad de Mutación vs Número de Iteraciones, variación de probabilidad de *mutación*

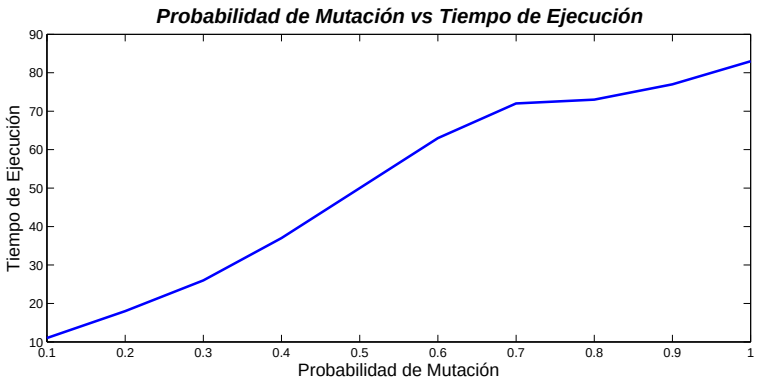


Figura 4.9: Probabilidad de Mutación vs Tiempos de Ejecución, variación de probabilidad de *mutación*

4.2. Soluciones al problema de Calendarización

4.2.1. Base de datos TOY

Se ha ejecutado **"agcbctt"** para la base de datos **Toy** del archivo *toy.txt*, ver Tabla 3.3, y se han obtenido los calendarios de las Figuras 4.10 y 4.11 que cumplen con todas las restricciones impuestas en la formulación **UD4**, ver Tabla 3.2, en su versión **ectt**.

Lo parámetros utilizados para este ejemplo fueron:

- **Tamaño de la población:** 60
- **Probabilidad de Cruce:** 0.7
- **Probabilidad de Mutación:** 0.4

	Currículum 1				
	Día 1	**Día 2**	**Día 3**	**Día 4**	**Día 5**
ranura 1	TecCos rB	ArcTec rB		TecCos rB	
ranura 2	TecCos rB			SceCosC rC	
ranura 3			SceCosC rC	ArcTec rB	TecCos rB
ranura 4			TecCos rB	ArcTec rB	SceCosC rC

Figura 4.10: Calendario para el Currículum 1, **Toy**

En la figura 4.12 se muestra la función de evaluación de la solución presentada.

	Currículum 2				
	Día 1	Día 2	Día 3	Día 4	Día 5
ranura 1					
ranura 2	Geotec rC	TecCos rB		TecCos rB	TecCos rB
ranura 3	TecCos rB	Geotec rC		Geotec rC	Geotec rC
ranura 4	TecCos rB			Geotec rC	

Figura 4.11: Calendario para el Currículum 2, **Toy**

```
>> [ffitnessv ffitness]=fitnessvector(tt,G,R,UC,C)
ffitnessv =
     0     0     0     0     0     0     0     0
ffitness =
     0
```

Figura 4.12: Aplicación de la función **fitness** a la base de datos **Toy**

El número de iteraciones que le tomó al **AG** encontrar una solución fue 22 y el tiempo de ejecución de 19 segundos.

Se puede encontrar más de una solución para este problema como la presentada en la página web del *SaTT: Scheduling and Timetabling Group*, ver Sección 3.2.1 y que se muestra en el Apéndice B.

La solución en Matlab® es una matriz de dimensión 40×8, ver Figura 4.13. El calendario para el *currículum* 1 y 2 se muestran de color azul y rojo respectivamente.

4.2.2. Un ejemplo con tres currículums

Se corrió un ejemplo adicional con tres currículums, ver Tabla C.1 en el Apéndice C, los resultados pueden leerse en las Figuras 4.14, 4.15 y 4.16, además la salidatesi de la función de **fitness**, con su valor óptimo se encuentra en la Figura 4.17.

Lo parámetros utilizados para este ejemplo fueron:

- **Tamaño de la población:** 60
- **Probabilidad de Cruce:** 0.8
- **Probabilidad de Mutación:** 0.4

1	1	1	3	3	2	50	40
2	2	1	2	2	2	50	42
3	3	1	0	0	0	0	0
4	4	1	3	3	2	50	40
5	5	1	0	0	0	0	0
6	1	2	3	3	2	50	40
7	2	2	0	0	0	0	0
8	3	2	0	0	0	0	0
9	4	2	1	1	3	40	30
10	5	2	0	0	0	0	0
11	1	3	0	0	0	0	0
12	2	3	0	0	0	0	0
13	3	3	1	1	3	40	30
14	4	3	2	2	2	50	42
15	5	3	3	3	2	50	40
16	1	4	0	0	0	0	0
17	2	4	0	0	0	0	0
18	3	4	3	3	2	50	40
19	4	4	2	2	2	50	42
20	5	4	1	1	3	40	30

21	1	1	0	0	0	0	0
22	2	1	0	0	0	0	0
23	3	1	0	0	0	0	0
24	4	1	0	0	0	0	0
25	5	1	0	0	0	0	0
26	1	2	4	4	3	40	18
27	2	2	3	3	2	50	40
28	3	2	0	0	0	0	0
29	4	2	3	3	2	50	40
30	5	2	3	3	2	50	40
31	1	3	3	3	2	50	40
32	2	3	4	4	3	40	18
33	3	3	0	0	0	0	0
34	4	3	4	4	3	40	18
35	5	3	4	4	3	40	18
36	1	4	3	3	2	50	40
37	2	4	0	0	0	0	0
38	3	4	0	0	0	0	0
39	4	4	4	4	3	40	18
40	5	4	0	0	0	0	0

Figura 4.13: Calendario en formato matriz, para la solución de **Toy**

	Currículum 1				
	Día 1	**Día 2**	**Día 3**	**Día 4**	**Día 5**
ranura 1	TecCos				TecCos
	rB				rB
ranura 2	SceCosC			ArcTec	TecCos
	rC		rC	rB	rB
ranura 3		ArcTec	ArcTec	SceCosC	
		rB	rB	rC	
ranura 4		TecCos	TecCos		
		rB	rB		

Figura 4.14: Calendario para el Currículum 1, ejemplo 2

	Currículum 2				
	Día 1	Día 2	Día 3	Día 4	Día 5
ranura 1	Geotec rC	TecCos rB		TecCos rB	Geotec rC
ranura 2	TecCos rB			Geotec rC	Geotec rC
ranura 3	TecCos rB		Geotec rC		TecCos rB
ranura 4					

Figura 4.15: Calendario para el Currículum 2, ejemplo 2

	Currículum 3				
	Día 1	Día 2	Día 3	Día 4	Día 5
ranura 1		SceCosC rC			
ranura 2			ArcTec rB		
ranura 3	SceCosC rC			ArcTec rB	
ranura 4				ArcTec rB	SceCosC rC

Figura 4.16: Calendario para el Currículum 3, ejemplo 2

```
>> [ffitnessv ffitness]=fitnessvector(tt,G,R,UC,C)
ffitnessv =
     0     0     0     0     0     0     0     0
ffitness =
     0
```

Figura 4.17: Aplicación de la función **fitness** a la base de datos del ejemplo 2

Capítulo 5

Conclusiones y Recomendaciones

5.1. Conclusiones

1. Los Métodos Metaheurísticos para el problema de Calendarización, específicamente los del sector educativo, son ampliamente utilizados, aunque se puede encontrar en la literatura aplicaciones que los combinan con métodos exactos tal como se puede ver en Oliveri [15]. Entre las Metaheurísticas más utilizadas están las que contienen memoria en sus procedimientos y guardan información de soluciones ya antes visitadas como los métodos **Tabú**, Jin-Kao [7]. Se ha podido encontrar también la aplicación de algoritmos evolutivos para este problema, muchas veces de tipo híbrido, Ki-Seok [19], así como del **Recocido Simulado** que ha sido utilizado en la solución de los datos **Toy** por Andrea Schaerf de la Universidad de Udine, Italia.

2. La inclusión de restricciones adicionales no es un problema hablando en términos computacionales. La dificultad solamente radica en el tiempo que tome programar el algoritmo que se diseñe para este propósito. En estainvestigación, por ejemplo, las restricciones *Currículum Compacto* y *Currículum Compacto versión 2* pudieron incluírse una o ambas en la función de evaluación. La formulación **UD** no considera a *Currículum Compacto versión 1*. Un problema que suele presentarse es la de disponibilidad de horarios por parte de los *profesores*, pero puede relacionarse con el grupo de restricciones de *No disponibilidad* presentada como *UNAVAILABILITY_CONSTRAINTS* en el formato de datos de entrada, ver Tabla 3.3.

3. El algoritmo de optimización sirve de base para que se diseñe y programe una interfaz gráfica para usuarios donde se muestre los recursos asignados y disponibles, con restricciones respectivas para reales, y que pueda mostrar calendarios filtrados para profesores, por materias y por aulas.

4. Para problemas de calendarización grandes el tiempo de ejecución puede incrementarse a valores no aceptables, es por esto que la aleatorización para construir los cromosoma de la población inicial podría sustituirse por algún método sistemático o por otra parte es posible generar soluciones no tan buenas mediante el uso de la programación lineal con restricciones relajadas. Este

tipo de mezcla de técnicas de solución es una alternativa digna de considerar, un ejemplo en problemas de calendarización deportiva se puede consultar en D. Oliveri [15].

5. Luego de probar en algunos ejemplos el **AG** diseñado, se concluye que las calibración de parámetros debe realizarse para cada problema particular, no solamente por el tamaño del mismo, sino por el tipo de formulación que se considere. Las diferentes restricciones imprimen dificultades distintas en cada caso.

5.2. Recomendaciones

1. Una de las cosas en las que se considera que un Algoritmo genético puede mejorar, es en la administración de la memoria, para guardar soluciones ya consideradas, tal como lo hacen los métodos **Tabú**, una referencia de cómo lograrlo pueden encontrarse en las aplicaciones que podrían darse del **Scatter Search**, Glover [6], método que además de (en alguna de sus versiones) considerar memoria, considera tamaños de *poblacion* más pequeños enfocándose fuertemente en la calidad de sus miembros, Laguna [13].

2. En esta implementación del procedimiento general del **AG** se considera un gran número de restricciones (8 en total, 4 duras y 4 suaves) que sin embargo pueden ampliarse. Por ejemplo, una restricción que no es considerada es la de distancia recorrida por los estudiantes cuando deben cambiar de aula entre clase y clase. El edificio donde se encuentran estos salones pueden ubicarse lejos una de la otra, de tal forma que se debería tomar en cuenta que a un estudiante le tomará un tiempo mayor que cero pasar a otro curso en este caso y no pueden programarse *lecturas* entre estos cambios.

3. La optimización de la programación de las actividades académicas de la ESPOL, pienso debería ser considerada con mayor énfasis por las instancias respectivas, al ser un conjunto de acciones que definitivamente disminuye las horas de trabajo de acoplamiento de horarios y aulas, así como reduce los gastos que podría demandar una planificación elemental de los recursos.

4. Se Sugiere estudiar la variante del problema de calendarización **ETT**, ver Sección 1.1, por el uso continuo que se podría dar en el **ICM** y en todas las unidades académicas de la **ESPOL**. La aplicación de un Algortimo Génetico para este caso se vuelve un poco más sencilla debido a que los estudiantes pueden ser considerados como entidades completas de estudio.

5. Un estudio sobre el número de estudiantes que toman los buses de la ESPOL en diferentes horas del día, sería importante para modelar una nueva restricción. Un problema que se identifica es el retraso de estudiantes a la primera hora, y una programación de clases que tome en consideración esta situación podría ayudar en algo este inconveniente.

Apéndice A

Programas adicionales

Programa 9 "gcrom.m": Genera un *cromosoma*

```
function cromosoma=gcrom(G,C,CR,RC,R,UC)
clc;
days=G(3,1);
periods=G(4,1);
rooms=G(2,1);
courses=G(1,1);
curricula=G(5,1);
le=days*periods*curricula;
de=le/curricula;
sr=size(R);
for i=1:le
    cromosoma(i,1)=i;
end
sum=0;
for j=1:le/days
    for i=1:days
        cromosoma(i+sum,2)=i;
    end
    sum=sum+days;
end
sumdos=0;
sumtres=0;
for k=0:(curricula-1)
    for i=1:periods
        for j=1:days
            cromosoma(j+sumtres+sumcuatro,3)=1+sumdos;
        end
        sumdos=sumdos+1;
        sumtres=sumtres+days;
    end
    sumdos=0;
end
cromosoma=generalecyprof(cromosoma,C,CR,RC,R,days,periods,curricula);
mejoraroom;
```

Programa 10 "roomycap.m": Ubica un *aula* y su *capacidad*

```
ss=size(R);
rr=ceil(ss(1)*rand);
fr=find(rr==RC(:,2));
if isempty(fr)==0
    while ismember(cromosoma(r+days*periods*(i-1),4),RC(fr,1))==1
        rr=ceil(ss(1)*rand);
        fr=find(rr==RC(:,2));
    end
    cromosoma(r+days*periods*(i-1),6)=R(rr,1);
    cromosoma(r+days*periods*(i-1),7)=R(rr,2);
 else
    cromosoma(r+days*periods*(i-1),6)=R(rr,1);
    cromosoma(r+days*periods*(i-1),7)=R(rr,2);
end
```

Programa 11 "mejoraroom.m": Mejora un *cromosoma* asignando nuevas *aulas*

```
suma=0;
for i=1:curricula
    for i=1:courses
        posm=[];
        posm=find(cromosoma((1+suma:de+suma),4)==i)+suma;
        for h=1:length(posm)
            rr=ceil(sr(1)*rand);
            cromosomamdos=cromosoma;
            cromosomados(posm(h),6)=R(rr,1);
            cromosomados(posm(h),7)=R(rr,2);
            fitmutadodos=fitness(cromosomamdos,G,R,UC,C);
            fitnormaldos=fitness(cromosoma,G,R,UC,C);
            if fitmutadodos<fitnormaldos
                cromosoma(posm(h),6)=R(rr,1);
                cromosoma(posm(h),7)=R(rr,2);
            end
        end
    end
    suma=suma+de;
end
```

Programa 12 "generalecyprof.m": Ubica una *lectura* y un *profesor*

```
function cromosoma=generalecyprof(cromosoma,C,CR,RC,R,days,periods,
curricula)
s=size(CR);
r=[];
for i=1:curricula
    memo=[];
    for j=3:s(2)
        curso=find(CR(i,j)==C(:,1));
        if isempty(find(CR(i,j)==C(:,1), 1))==0
            lec=C(curso,3);
            for k=1:lec
                %r=round(rand*(days*periods-1)+1);
                r=ceil(days*periods*rand);
                if ismember(r,memo)
                    while ismember(r,memo)
                        r=ceil(days*periods*rand);
                    end
                        cromosoma(r+days*periods*(i-1),4)=curso;
                        cromosoma(r+days*periods*(i-1),5)=C(curso,2);
                        mem(k)=r;
                        roomycap;
                        cromosoma(r+days*periods*(i-1),8)=C(curso,5);
                else
                    cromosoma(r+days*periods*(i-1),4)=curso;
                    cromosoma(r+days*periods*(i-1),5)=C(curso,2);
                    mem(k)=r;
                    roomycap;
                    cromosoma(r+days*periods*(i-1),8)=C(curso,5);
                end
                memo=union(memo,mem);
            end
        end
    end
end
```

Programa 13 "roomoccupancy.m": Procedimiento para *Aulas ocupadas*

```
s=size(R);
dp=le/curricula;
cur=[];
suma=0;
for i=1:dp
    suma=0;
    for j=1:curricula
        cur(i,j)=cromosoma(i+suma,6);
        suma=suma+dp;
    end
    if sum(cur(i,:))==0
        cur(i,curricula+1)=0;
    else
        for k=1:s(1)
            c=find(cur(i,:)==k);
            if c>=0
                prev(k)=length(c)-1;
            else
                prev(k)=0;
            end
            cur(i,curricula+1)=sum(prev);
        end
    end
end
froomoccupancy=sum(cur(:,curricula+1));
```

Programa 14 "conflicts.m": Procedimiento para *Conflictos*

```
dp=le/curricula;
cur=[];
suma=0;
for i=1:dp
    suma=0;
    for j=1:curricula
        cur(i,j)=cromosoma(i+suma,5);
        suma=suma+dp;
    end
    if sum(cur(i,:))==0
        cur(i,curricula+1)=0;
    else
        c=find(cur(i,:)~=0);
        d=[];
        for k=1:length(c)
            d(k)=cur(i,c(k));
            dset=unique(d);
            cur(i,curricula+1)=length(c)-length(d);
        end
    end
end
fconflicts=sum(cur(:,curricula+1));
```

Programa 15 "uconstraints.m": Procedimiento para *Disponibilidad*

```
s=size(UC);
for i=1:s(1)
    uco=find(cromosoma(:,2)==UC(i,2) & cromosoma(:,3)==UC(i,3));
    for j=1:length(uco)
        if UC(i,1)==cromosoma(uco(j),4)
            cromosoma(uco(j),9)=1;
        else
            cromosoma(uco(j),9)=0;
        end
    end
end
%cromosoma;
fuconstraints=sum(cromosoma(:,9));
```

Programa 16 "roomcapacity.m": Procedimiento para *Capacidad de aulas*

```
cromosoma(:,9)=[];
for i=1:le
    if cromosoma(i,7)<cromosoma(i,8)
        cromosoma(i,9)=cromosoma(i,8)-cromosoma(i,7);
    else
        cromosoma(i,9)=0;
    end
end
cromosoma;
froomcapacity=sum(cromosoma(:,9));
```

Programa 17 "minwdays.m": Procedimiento para *Días de trabajo mínimo*

```
cromosoma(:,9)=[];
s=size(C);
de=le/curricula;
suma=0;
for j=1:curricula
    for k=1:s(1)
        for i=1:de
            if cromosoma(i+suma,4)==C(k,1)
                cc(i,k)=cromosoma(i,2);
            else
                cc(i,k)=0;
            end
        end
        if length(setdiff(unique(cc(:,k)),0))<C(k,4)
           && sum(unique(cc(:,k)))~=0
            di=C(k,4)-length(setdiff(unique(cc(:,k)),0));
            vec(k,j)=di;
        end
        if sum(unique(cc(:,k)))==0
            vec(k,j)=0;
        end
    end
    cc=[];
    suma=suma+dp;
end
fminwdays=sum(sum(vec));
```

Programa 18 "ccompactness.m": Procedimiento para *Currículum compacto*

```
de=le/curricula;
ccomp=0;
suma=0;
for j=1:curricula
    pos=[];
    vec=[];
    for k=1:days
        pos=find(cromosoma(1:de,2)==k)+suma;
        for i=1:periods
            if cromosoma(pos(i),4)~=0
                vec(i,k)=cromosoma(pos(i),3);
            else
                vec(i,k)=0;
            end
        end
        f=find(vec(:,k)~=0);
        if length(f)>1
            for m=1:length(f)-1
                if f(m+1)-f(m)>1
                    ccomp=ccomp+1;
                end
            end
        end
    end
    suma=suma+de;
end
fccompactness=ccomp;
```

Programa 19 "rstability.m": Procedimiento para *Estabilización de aulas*

```
de=le/curricula;
suma=0;
sumados=0;
for j=1:curricula
    vec=[];
    for k=1:courses
        pos=find(cromosoma(1+suma:de+suma,4)==k)+suma;
        if isempty(pos)==0
            for h=1:length(pos)
                vec(h,k)=cromosoma(pos(h),6);
            end
            rs=length(unique(vec(:,k)))-1;
            sumados=sumados+rs;
        end
    end
suma=suma+de;
end
frstability=sumados;
```

Programa 20 "cruce.m" Parte I: Proceso de cruce de un par de *cromosomas*

```
function [hijo hija]=cruce(padre,madre,G)
days=G(3,1);
periods=G(4,1);
rooms=G(2,1);
courses=G(1,1);
curricula=G(5,1);
le=days*periods*curricula;
de=le/curricula;
hijo=padre;
hijo(:,4:8)=0;
hija=hijo;
suma=0;
for j=1:curricula
    for i=1:courses
        posp=[];
        posm=[];
        posp=find(padre((1+suma:de+suma),4)==i)+suma;
        posm=find(madre((1+suma:de+suma),4)==i)+suma;
        for k=1:length(posp)
            fc=[];
            r=rand;
            if r>0.5
                if hijo(posp(k),4)==0
                    hijo(posp(k),4:8)=padre(posp(k),4:8);
                    if hija(posm(k),4)==0
                        hija(posm(k),4:8)=madre(posm(k),4:8);
                    else
                        fc=[];
                        fc=find(hija(1+suma:de+suma,4)==0)+suma;
                        pp=ceil(length(fc)*rand());
                        hija(fc(pp),4:8)=madre(posm(k),4:8);
                    end
                else
                    if hijo(posm(k),4)==0
                        hijo(posm(k),4:8)=padre(posp(k),4:8);
                        if hija(posp(k),4)==0
                            hija(posp(k),4:8)=madre(posm(k),4:8);
```

Programa 21 "cruce.m" Parte II: Proceso de cruce de un par de *cromosomas*

```
                    else
                        fc=[];
                        fc=find(hija(1+suma:de+suma,4)==0)+suma;
                        pp=ceil(length(fc)*rand());
                        hija(fc(pp),4:8)=madre(posm(k),4:8);
                    end
                else
                    fc=[];
                    fc=find(hijo(1+suma:de+suma,4)==0)+suma;
                    pp=ceil(length(fc)*rand());
                    hijo(fc(pp),4:8)=padre(posp(k),4:8);
                    if hija(fc(pp),4)==0
                        hija(fc(pp),4:8)=madre(posm(k),4:8);
                    else
                        fc=[];
                        fc=find(hija(1+suma:de+suma,4)==0)+suma;
                        pp=ceil(length(fc)*rand());
                        hija(fc(pp),4:8)=madre(posm(k),4:8);
                    end
                end
            end
        else
            if hijo(posm(k),4)==0
                hijo(posm(k),4:8)=madre(posm(k),4:8);
                if hija(posp(k),4)==0
                    hija(posp(k),4:8)=padre(posp(k),4:8);
                else
                    fc=[];
                    fc=find(hija(1+suma:de+suma,4)==0)+suma;
                    pp=ceil(length(fc)*rand());
                    hija(fc(pp),4:8)=padre(posp(k),4:8);
                end
            else
                if hijo(posp(k),4)==0
                    hijo(posp(k),4:8)=madre(posm(k),4:8);
                    if hija(posm(k),4)==0
                        hija(posm(k),4:8)=padre(posp(k),4:8);
```

Programa 22 "cruce.m" Parte III: Proceso de cruce de un par de *cromosomas*

```
                        else
                            fc=[];
                            fc=find(hija(1+suma:de+suma,4)==0)+suma;
                            pp=ceil(length(fc)*rand());
                            hija(fc(pp),4:8)=padre(posp(k),4:8);
                        end
                    else
                        fc=[];
                        fc=find(hijo(1+suma:de+suma,4)==0)+suma;
                        pp=ceil(length(fc)*rand());
                        hijo(fc(pp),4:8)=madre(posm(k),4:8);
                        if hija(fc(pp),4)==0
                            hija(fc(pp),4:8)=padre(posp(k),4:8);
                        else
                            fc=[];
                            fc=find(hija(1+suma:de+suma,4)==0)+suma;
                            pp=ceil(length(fc)*rand());
                            hija(fc(pp),4:8)=padre(posp(k),4:8);
                        end
                    end
                end
            end
        end
    end
    suma=suma+de;
end
```

Programa 23 "crucepoblacion.m": Selecciona a un padre y una madre para cruzarlos y elige quien formará parte de la nueva generación

```
function poblacion=crucepoblacion(poblacion,G,R,UC,C,probcruce)
days=G(3,1);
periods=G(4,1);
rooms=G(2,1);
courses=G(1,1);
curricula=G(5,1);
le=days*periods*curricula;
de=le/curricula;
s=size(poblacion);
for i=1:s(3)
    r1=ceil(s(3)*rand);
    r2=ceil(s(3)*rand);
    r=rand;
    if r<=probcruce
        padre=poblacion(:,:,r1);
        madre=poblacion(:,:,r2);
        if padre==madre
            poblacionbandera(:,:,i)=padre;
        else
            [hijo hija]=cruce(padre,madre,G);
            fhijo=fitness(hijo,G,R,UC,C);
            fhija=fitness(hija,G,R,UC,C);
            if fhijo<=fhija
                poblacionbandera(:,:,i)=hijo;
            else
                poblacionbandera(:,:,i)=hija;
            end
        end
    else
        poblacionbandera(:,:,i)=poblacion(:,:,i);
    end
end
poblacion=poblacionbandera;
```

Programa 24 "mutacion.m": Ubica un gen en un *periodo* disponible

```
function cromosoma=mutacion(cromosoma,G,R,UC,C,probmutacion)
days=G(3,1);
periods=G(4,1);
rooms=G(2,1);
courses=G(1,1);
curricula=G(5,1);
le=days*periods*curricula;
de=le/curricula;
sr=size(R);
suma=0;
for i=1:curricula
    for i=1:courses
        posm=[];
        posm=find(cromosoma((1+suma:de+suma),4)==i)+suma;
        for k=1:length(posm)
            r=rand;
            if r<probmutacion
                fm=[];
                fm=find(cromosoma(1+suma:de+suma,4)==0)+suma;
                pm=ceil(length(fm)*rand());
                cromosomamut=cromosoma;
                cromosomamut(fm(pm),4:8)=cromosomamut(posm(k),4:8);
                cromosomamut(posm(k),4:8)=0;
                fitmutado=fitness(cromosomamut,G,R,UC,C);
                fitnormal=fitness(cromosoma,G,R,UC,C);
                if fitmutado<fitnormal
                    cromosoma(fm(pm),4:8)=cromosoma(posm(k),4:8);
                    cromosoma(posm(k),4:8)=0;
                end
            end
        end
    end
suma=suma+de;
end
```

Programa 25 "mutacionpoblacion.m": Define si un cromosoma entra al proceso de *mutación*

```
function poblacion=mutacionpoblacion(poblacion,G,R,UC,C,probmutacion)
days=G(3,1);
periods=G(4,1);
curricula=G(5,1);
le=days*periods*curricula;
de=le/curricula;
s=size(poblacion);
for i=1:s(3)
    r=rand;
    if r<probmutacion
        poblacion(:,:,i)=mutacion(poblacion(:,:,i),G,R,UC,C
                        ,probmutacion);
    end
end
```

Apéndice B

Calendarios adicionales

	Currículum 1				
	Día 1	**Día 2**	**Día 3**	**Día 4**	**Día 5**
ranura 1		ArcTec room: rB		SceCosC room: rB	
ranura 2		TecCos room: rB	ArcTec room: rB	TecCos room: rB	
ranura 3	ArcTec room: rB	SceCosC room: rC	TecCos room: rB		TecCos room: rB
ranura 4	SceCosC room: rC				TecCos room: rB

Figura B.1: Calendario para el Currículum 1, **SaTT**, **Toy**

	Currículum 2				
	Día 1	**Día 2**	**Día 3**	**Día 4**	**Día 5**
ranura 1		Geotec room: rC			
ranura 2	Geotec room: rC	TecCos room: rB	Geotec room: rC	TecCos room: rB	
ranura 3	Geotec room: rC		TecCos room: rB	Geotec room: rC	TecCos room: rB
ranura 4					TecCos room: rB

Figura B.2: Calendario para el Currículum 2, **SaTT**, **Toy**

Apéndice C

Datos de entrada para ejemplo de tres currículums

Tabla C.1: Base de datos para ejemplo de tres currículums

```
Name: Toy
Courses: 4
Rooms: 3
Days: 5
Periods_per_day: 4
Curricula: 3
Constraints: 8

COURSES:
SceCosC Ocra 3 3 30
ArcTec Indaco 3 2 42
TecCos Rosa 5 4 40
Geotec Scarlatti 5 4 18

ROOMS:
rA 32
rB 50
rC 40

CURRICULA:
Cur1 3 SceCosC ArcTec TecCos
Cur2 2 TecCos Geotec
Cur3 2 SceCosC ArcTec

UNAVAILABILITY_CONSTRAINTS:
TecCos 2 0
TecCos 2 1
TecCos 3 2
TecCos 3 3
ArcTec 4 0
ArcTec 4 1
ArcTec 4 2
ArcTec 4 3

ROOM_CONSTRAINTS:
SceCosC rA
Geotec rB
TecCos rC

END.
```

Bibliografía

[1] Fabio De Cesco, Andrea Schaerf, and Luca Di Gaspero. Benchmarking curriculum-based course timetabling: Formulations, data formats, instances, validation, and results. 2008.

[2] Lance D. Chambers. *Practical Handbook of GENETIC ALGORITHMS, Volumen III*. CRC Press LLC, 1999.

[3] Chin-Yen Chen. Using integer programming to solve the school timetabling problem at chin-min institute of technology. *American Academy of Business*, 2008.

[4] Adenso Díaz. *Optimizaci?n Heur?stica y Redes Neuronales*. Addison Wesley, 1996.

[5] Molina Juan Enrique. Algoritmos evolutivos para la resolución de un problema del tipo timetabling. *Memoria para el título de Ingeniero en Inform?tica Aplicada*, 2007.

[6] Fred Glover and Gary A. Kochenberger. *Handbook of Methaheuristics*. Kluwers' International Series in Operation Research and Management Science.

[7] Jin-Kao Hao and Zhipeng Lu. Adaptive tabu search for course timetabling. *ELSEVIER*, 2008.

[8] J.H. Holland. Adaptation in natural and artificial systems. *University of Michigan Press, Ann Arbor, Michigan; re-issued by MIT Press (1992)*, 1975.

[9] Milena Karova, Vassil Smarkov, and Stoyan Penev. Genetic operators crossover and mutation in solving the tsp problem. *International Conference on Computer Systems and Technologies*, 2005.

[10] J.H. Kingston, R.F. Weare, E.K. Bueke, and K.S. Jackson. Automated timetabling: The state of the art. *The Computer Journal, Volume 40(9)*, 1997.

[11] Joseph Y-T Leung. *Handbook of Scheduling. Algorithms, Lodels and Perfomance Analysis*. Chapman & Hall/CRC. Computer and Information Science Series, 2004.

[12] Concepción Maroto Álvarez and Javier Alcaraz Soria. Genetic algorithms for the resource-constrained project scheduling problem (rcpsp). *Boletín de Estadística e Investigaci?n Operativa*, 2009.

[13] Manuel Laguna Rafael Martí. *Scatter Search: Dise?o Básico y Estrategias Avanzadas*.

[14] Luis Rodríguez Ojeda. *Matlab® Conceptos Básicos y Programación*. Instituto de Ciencias Matem?ticas, Escuela Superior Polit?cnica del Litoral, 2007.

[15] D. Oliveri and F. Della Croce. Scheduling the italian football league: an ilp-based approach. *ELSEVIER*, 2004.

[16] Flores Pedro, Brau Ernesto, Moteverde Jazm?n, Salazar Norman, Figueroa José, Cadena Eliseo, and Lizárraga Caleb. Experimentos con algoritmos genéticos para resolver un problema real de programación maestros-horarios-cursos. *Departamento de Matem?ticas Universidad de Sonora, Hermosillo Sonora CP 83000 México*.

[17] Andrea Schaerf, Fabio De Cesco, and Barry McCollum. The second international timetabling competition (itc-2007): Curriculum-based course timetabling (track 3). *Technical Report*, 2007.

[18] Áslaug Sóley Bjarnadóttir. Solving the vehicle routing problem with genetic algorithms. *Informatics and Mathematical Modelling, IMM*, 2004.

[19] Ki-Seok Sung and Enzhe Yu. A genetic algorithm for a university weekly courses timetabling problem. *Blackwell Publishers*, 2007.