Universidad Nacional Autónoma de México

Facultad de Ingeniería

SAG-FI: Sistema de Administración de Grupos para la Facultad de Ingeniería

T E S I S

Que para Obtener el Título de

Ingeniero en Computación

Presenta

Alejandro Cedeño Quintero

Director

Ing. Stalin Muñoz Gutiérrez

Ciudad Universitaria, 2013

A mis padres, José Luis Valeriano y Alejandrina

A mis hermanos, Samantha y José Luis

Agradecimientos

A grupo LINDA: Stalin, Luis Sergio, Genaro, Rigoberto y Marco

A mi amigo Abraham

"El progreso es imposible sin el cambio; y aquellos que no pueden cambiar de opinión no pueden cambiar nada."

George Bernard Shaw

Índice general

Índice de figuras

Índice de cuadros

Índice de algoritmos

Acrónimos

CACEI Consejo de Acreditación de la Enseñanza de la Ingeniería.

CSP Constraint Satisfaction Problems.

DIE División de Ingeniería Eléctrica.

DLA Diffusion-Limited Aggregation.

FI Facultad de Ingeniería.

HS Harmony Search.

ITC International Timetabling Competition.

JSON JavaScript Object Notation.

LINDA Laboratorio de Investigación para el Desarrollo Académico.

MTD Máquina de Turing Determinista de una cinta.

MTND Máquina de Turing No-Determinista de una cinta.

NSGA-II Non-dominated Sorting Genetic Algorithm-II.

PATAT Practice and Theory of Automated Timetabling.

SA Simulated Annealing.

SAG-FI Sistema de Administración de Grupos para la Facultad de Ingeniería.

SA-SWO Simulated Annealing - Squeaky Wheel Optimization.

SWO Squeaky Wheel Optimization.

TS Tabu Search.

UNAM Universidad Nacional Autónoma de México.

Prólogo

La asignación de horarios o *timetabling* es un problema que está presente en instituciones educativas, de salud, de transporte, entre otras. Podría decirse que en cualquier lugar donde se tengan que calendarizar tareas, clases, exámenes, turnos de empleados y hasta actividades deportivas, siempre buscando la mejor manera de distribuir el tiempo y recursos disponibles en cierto número de eventos, cumpliendo condiciones cuya naturaleza caracteriza al *timetabling* como un problema difícil de resolver, tanto para las personas como para las computadoras, ya que encontrar una respuesta satisfactoria no siempre puede ser hallada en un tiempo razonable si el espacio de búsqueda es enorme.

Por años se han desarrollado técnicas e investigaciones multidisciplinarias que buscan ofrecer soluciones de calidad para distintos tipos de calendarizaciones, algunas más efectivas que otras para ciertos casos, pero no hay garantía de que exista un algoritmo que pueda manejar la complejidad de todos los casos en un tiempo aceptable.

El trabajo aquí presentado se enfoca en ofrecer una herramienta computacional para facilitar la asignación de grupos en la Facultad de Ingeniería (FI) de la Universidad Nacional Autónoma de México (UNAM) donde las diversas asignaturas, grupos, aulas y profesores han obligado a depender de asignaciones anteriores y a hacer mínimas modificaciones, ya que cambios radicales pueden llevar a resultados peores en ocasiones difíciles de corregir. [1]

[1] Entrevista con la secretaria académica de la División de Ingeniería Mecánica e Industrial de la FI, Ing. Miriam G. Mendoza Cano, noviembre 2011.

Capítulo 1

Introducción

1.1. Análisis de Algoritmos

Informalmente, un **algoritmo** es una secuencia finita de operaciones cuyo propósito es resolver un tipo específico de problema y posee cinco características fundamentales:

1. siempre debe terminar en un número finito de pasos,

2. cada paso debe ser definido con precisión,

3. tiene cero o más entradas representadas por cantidades especificadas antes de que el algoritmo comience o a medida que se ejecuta,

4. tiene una o más salidas, las cuales son cantidades que guardan una relación específica con las entradas, y

5. se espera que sea eficaz, en el sentido de que todas sus operaciones sean lo suficientemente sencillas para que una persona pueda realizarlas, en tiempo finito, utilizando solamente lápiz y papel [Knu97].[2]

[2]La conexión entre la notación informal de algoritmo y su definición precisa ha llegado a llamarse la *Tesis Church-Turing* [Sip13], de la cual se hablará más adelante.

Cuando se tienen dos o más algoritmos que pueden aplicarse al mismo problema, es necesario establecer medidas que permiten compararlos y ofrezcan métricas objetivas para tomar decisiones. El *Análisis de Algoritmos* o *Análisis Algorítmico* es el campo dedicado a la especificación de las características de desempeño de los algoritmos [Knu97]. Existen dos tipos principales de enfoques que se utilizan para determinar dichas características y compararlos: el primero de ellos es el *benchmarking* (comparación empírica) que consiste en ejecutar, en una computadora específica, el programa que implementa un algoritmo para medir su tiempo de ejecución y la memoria consumida; por otro lado, el segundo enfoque consiste en analizar el número de pasos y el espacio requerido por el algoritmo independiente de la implementación [Rus03]. El primero de los enfoques será abordado en el *Capítulo 4. Resultados y observaciones*, en esta sección se dará una introducción del segundo.

Para efectuar dicho análisis, primero se debe realizar una abstracción sobre la entrada, es decir, encontrar uno o más parámetros que caractericen el tamaño de ésta, para después, realizar otra abstracción pero ahora sobre la implementación con el fin de encontrar alguna medida que refleje el tiempo que tarda en ejecutarse el algoritmo, sin que este ligado a algún compilador o computadora en particular. A esta caracterización se le llama $\mathbf{T(n)}$, donde n denota el tamaño de la entrada y $T(n)$ el tiempo de ejecución que toma el algoritmo para dicha entrada.

Existen principalmente dos problemas que complican el análisis de un algoritmo utilizando esta caracterización: el primero trata sobre la dificultad de encontrar un parámetro n que caracterice su número de pasos. Lo mejor que se puede hacer en esta situación, es calcular el caso menos favorable $T_{worst}(n)$ o el caso promedio $T_{avg}(n)$ y utilizar notaciones, como la **notación-O**, para describir el tiempo asintótico de ejecución en términos de funciones cuyos dominios son el conjunto de números naturales[3] [Cor09]. El segundo problema es que los

[3]Existen otro tipo de notaciones como la notación-Θ, Ω, o y ω cada una con un significado preciso para cierto tipo de análisis.

algoritmos no pueden analizarse de manera exacta, por lo que se recurre a aproximaciones [Rus03].

El objetivo principal del análisis es conocer cómo el tiempo de ejecución escala con el tamaño de la entrada, esto es llamado **análisis asintótico** e ignora términos de bajo orden y factores constantes, concentrándose en la forma de la curva del tiempo de ejecución. Por ejemplo, usando el siguiente algoritmo [Rus03]:

Algoritmo 1 Suma de una secuencia de números

Entrada:
A : *Una secuencia de números*
Salida:
B : *El resultado de la suma de los elementos de A*
1: $B \leftarrow 0$
2: **for** $i = 1 \rightarrow tamaño(A)$ **do**
3: $\quad B \leftarrow B + A[i]$
4: **end for**
5: **return** B

Para caracterizar el número de pasos se cuentan las líneas de código que se ejecutan dada una entrada n que representa el número de elementos de la secuencia. Cuando n es igual a 1 se ejecutan las líneas 1, 2, 3 y 5 [4]; cuando n es igual a 3, se ejecutan las líneas 1, 2, 3, 2, 3, 2, 3 y 5. Este comportamiento puede modelarse como $T(n) = 2n + 2$. Para analizar el desempeño de algoritmos tomando en cuenta el peor de los escenarios posibles, se utilizará la *notación-O* la cual es usada cuando solamente se tiene el límite asintótico superior [Cor09], en este caso el valor de n, así este caso particular puede ser expresado como $O(n)$ que significa la existencia de una relación lineal entre el tiempo de ejecución y el consumo de memoria.

Al generalizar el ejemplo anterior, para cualquier caso se tiene:

Definición 1. [Rus03] $T(n)$ es $O(g(n))$ si $T(n) \leq cg(n)$ para alguna c, para toda $n > n_0$.

[4]La línea 2 y 4 del código se toman como una sola debido a que forman la misma instrucción.

Definición 2. [Cor09] $O(g(n)) = \{f(n)$: existen constantes positivas c y n_0 tal que $0 \leq f(n) \leq cg(n)$ para toda $n \geq n_0\}$.[5]

La *notación-O* al no considerar factores constantes se vuelve más fácil de usar pero menos precisa que $T(n)$, por lo que se establece un equilibrio entre precisión y facilidad de análisis. La figura 1.1 muestra algunos ejemplos para comprender mejor ambas definiciones.

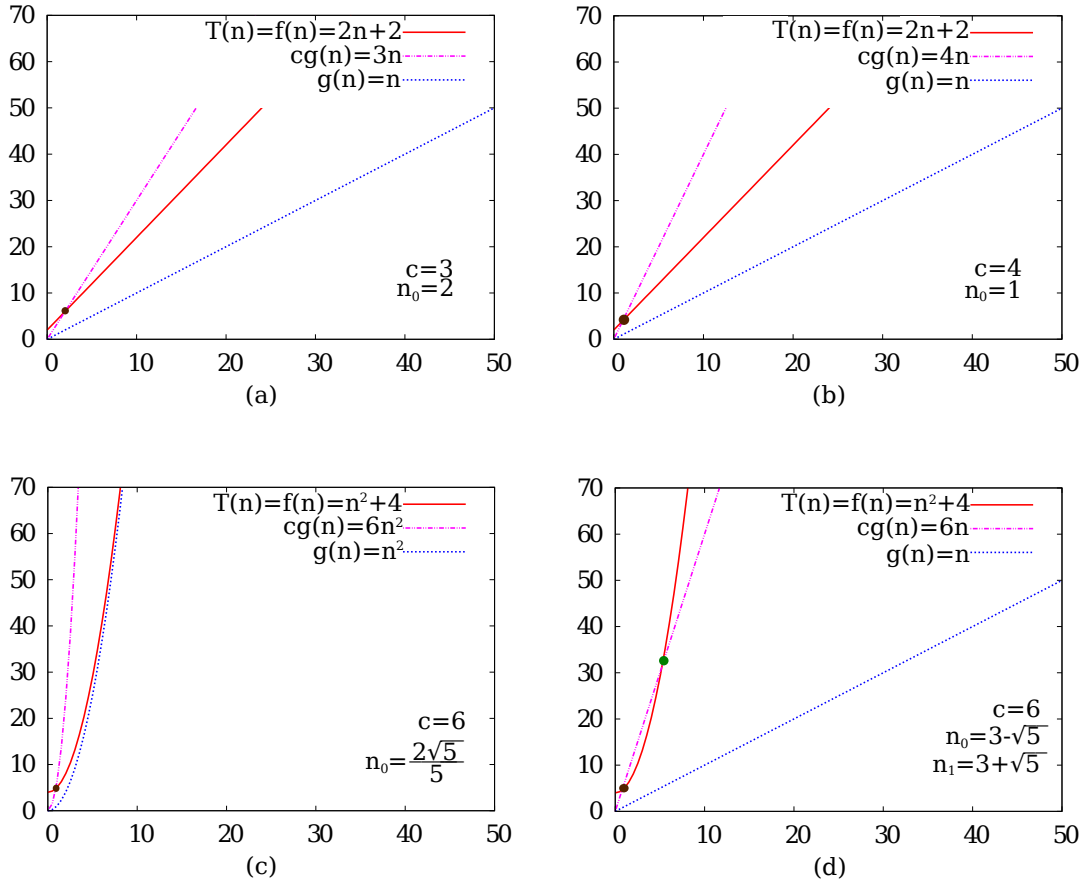

Figura 1.1: Comparativa entre f(n) y g(n). En la figura (a) existe una constante $c = 3$ que al multiplicar a $g(n)$ cruza a la recta $f(n)$ (equivalente a $T(n)$ la complejidad del Algoritmo 1) y satisfacen las definiciones anteriores, esta intersección sucede en n_0 igual a 2 (la abscisa del punto de cruce); en la figura (b) se presenta el mismo caso pero con una c distinta, por ello ahora su valor es igual a 4 y n_0 es igual a 1; la figura (c) presenta a $g(n)$ y $T(n)$ como funciones cuadráticas, de igual manera existe c y n_0; finalmente en la figura (d) $f(n)$ es una función cuadrática y $g(n)$ es una función lineal, aunque no existe una constante c que cumpla las condiciones, para cualquier $c > 1$, $g(n)$ supera a $T(n)$ en un intervalo a partir de n_0, pero también existe un n_1 tal que si $n > n_1$ entonces $cg(n) < f(n)$.

[5]En notación de conjuntos, los dos puntos deben de leerse como "tal que".

1.2. Análisis de Complejidad

El campo del *Análisis de Complejidad* se encarga de estudiar y clasificar problemas en lugar de algoritmos [Rus03]. Cuando dichos problemas son formulados como *problemas de decisión* (preguntas o funciones cuya solución es expresada como un "sí" o un "no"), pueden clasificarse en distintas *clases de complejidad*, definidas como el conjunto de funciones que pueden ser calculadas dentro del dominio de los recursos especificados [Aro09].

Existe una gran división entre los problemas que pueden ser resueltos en tiempo polinómico[6] y los que no, sin importar que algoritmo utilicen [Rus03]. Un **algoritmo de tiempo polinómico** es aquel que está definido por una función de complejidad en el tiempo $O(p(n))$ para alguna función polinómica $p(n)$, donde n es usada para denotar el tamaño de la entrada; cualquier otro algoritmo cuya función de complejidad en el tiempo $O(f(n))$ para alguna función no-polinómica $f(n)$, tenga un comportamiento exponencial respecto al tamaño de entrada n, es llamado **algoritmo de tiempo exponencial** [Gar79].[7]

La complejidad en el tiempo de ambos tipos de algoritmos permite clasificar los problemas en **tratables**, cuando pueden ser eficientemente resueltos, e **intratables**, cuando pueden ser resueltos, pero no lo suficientemente rápido para poder emplear su solución [Hop01]. Generalmente la primera clasificación es asociada a los *algoritmos de tiempo polinómico* y la segunda a los *algoritmos de tiempo exponencial*, aunque puede haber excepciones que dependan de información adicional. [8]

La clase de problemas polinómicos, los cuales pueden ser resueltos en tiempo $O(n^k)$ para alguna k es llamada P. Otra clase importante de problemas es la NP, la clase de problemas

[6] El término inglés *polynomial* puede encontrarse en la literatura en español como *polinómico* o *polinomial* aunque ambos términos no son reconocidos por la Real Académica Española. Otra traducción puede ser *expresado por un polinomio*.

[7] Algunas funciones, como $n^{\log n}$, típicamente no se consideran como exponenciales [Gar79].

[8] Por ejemplo, si se tuvieran dos algoritmos diferentes, el primero de ellos con complejidad $O(n^{20})$ y el segundo con complejidad $O(1.1^n)$, y con ellos se buscará resolver un problema dado en el menor tiempo posible para una entrada de tamaño pequeño, sería conveniente utilizar el segundo algoritmo, que a pesar de ser de tiempo exponencial, resolvería más rápidamente el problema que si se empleara el primer algoritmo.

polinómicos no determinísticos. Un problema pertenece a esta clase si se tiene algún algoritmo que supone una solución y puede comprobar si dicha conjetura es correcta en un tiempo polinómico.[9] Una de las grandes preguntas abiertas en la ciencia de la computación es si la clase NP es igual a la clase P, muchos científicos de la computación están convencidos de que son diferentes, que los problemas NP son inherentemente difíciles y no tienen algoritmos de tiempo polinómico, pero esto no ha sido posible probarlo formalmente.[10]

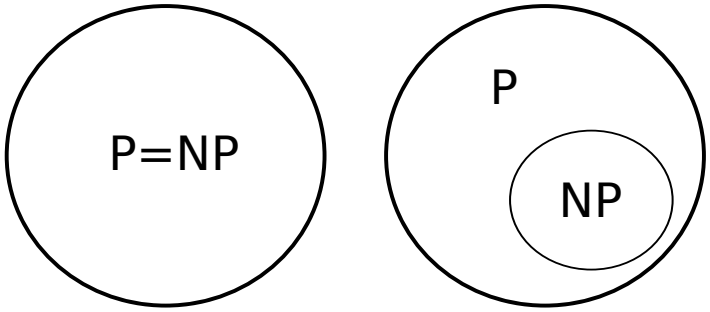

Figura 1.2: Posibles relaciones entre las clases P y NP. La figura de la izquierda muestra que la clase P es equivalente a la clase NP; la figura de la derecha muestra a la clase P contenida dentro de la clase NP [Sip13].

1.3. Máquina de Turing

La formalización de los conceptos de *algoritmo* y de las *clases* P y NP pueden relacionarse con la **Máquina de Turing**, la cual consiste en un modelo computacional propuesto por Alan M. Turing en 1936 [Tur37], similar a un *autómata finito* pero con memoria ilimitada y sin restricciones [Sip13]. En particular, es el modelo más utilizado debido a la similitud que tiene con una computadora, ya que ambos pueden aceptar los mismos tipos de lenguajes: *los*

[9] Existen numerosas clases de complejidad, algunas más específicas que otras para ciertos tipos de problemas. "Complexity Zoo" es una referencia en línea que brinda una introducción a muchas de ellas. `https://complexityzoo.uwaterloo.ca/Complexity_Zoo`

[10] "P vs NP Problem", `http://www.claymath.org/millennium/P_vs_NP`

lenguajes recursivamente numerables.[11]

Definición 3. [Sip13] Una Máquina de Turing es una 7-tupla

$$(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$$

donde Q, Σ, Γ son todos conjuntos finitos y

1. Q es un conjunto de estados,
2. Σ es el alfabeto de entrada que no incluye el símbolo de vacío $\sqcup$,
3. Γ es el alfabeto de la cinta, donde $\sqcup \in \Gamma$ y $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ es la función de transición,
5. $q_0 \in Q$ es el estado inicial,
6. $q_{accept} \in Q$ es el estado de aceptación, y
7. $q_{reject} \in Q$ es el estado de rechazo, donde $q_{reject} \neq q_{accept}$.

Una Máquina de Turing es representada mediante una cinta infinita dividida en cuadros o casillas donde se escriben, borran o modifican símbolos definidos por su alfabeto. Estas operaciones son realizadas a través de una cabeza lectora que se sitúa sobre alguna casilla de la cinta y cuyo movimiento es bidireccional (izquierda y derecha).[12]

[11] El término "recursivamente numerables" proviene de los formalismos computacionales que son anteriores a la Máquina de Turing los cuales definen la misma clase de lenguajes que pueden listar en cierto orden todos sus elementos, es decir, pueden "numerarlos". En este contexto, el término "recursivo" es sinónimo de "decidible". Si se considera un lenguaje L como un "problema", el problema L es "decidible" si L es un lenguaje recursivo y puede ser resuelto efectivamente por un algoritmo; es "semi-decidible" si existe un algoritmo que pueda numerar los elementos de L y que además puede ejecutarse por un tiempo indeterminado; y es "indecidible" si L no es un lenguaje recursivo [Hop01].

[12] En las siguientes secciones se mostrarán representaciones esquemáticas de dos tipos de Máquinas de Turing.

Existen diferentes variantes de la Máquina de Turing y tanto éstas como el modelo original reconocen la misma clase de lenguajes, su invariancia a ciertos cambios describe a estos modelos como robustos [Sip13]. Estos modelos aparecen en el tiempo en que se buscaba formalizar la notación de *efectivamente computable*, de manera que permitiera distinguir entre lo computable y lo no-computable. Varios formalismos alternativos fueron evolucionando en el intento de concretar esta notación. Entre ellos:

- Máquinas de Turing (Alan Turing),
- Sistemas Post (Emil Post),
- Funciones μ-recursivas (Kurt Gödel, Jacques Herbrand),
- Cálculo-λ (Alonzo Church, Stephen C. Kleene),
- Lógica Combinatoria (Moses Schönfinkel, Haskell B. Curry) [Koz97], y
- Autómatas Celulares (Stephen Wolfram) [Wol02].

Todos estos sistemas incorporaban la idea de *efectivamente computable* de una manera u otra ya que trabajaban sobre distintos tipos de datos; por ejemplo, las Máquinas de Turing manipulan cadenas sobre un alfabeto finito, las funciones μ-recursivas lo hacen sobre los números naturales y el cálculo-λ lo hace sobre términos-λ, sin embargo, pueden establecerse traducciones entre estos diferentes tipos de datos. Por ejemplo, existe una correspondencia uno-a-uno entre cadenas en $\{0,1\}^*$ y el conjunto de números naturales $\mathbb{N} = \{0,1,2,\dots\}$ definida por:

$$x \to \#(1x) - 1 \tag{1.1}$$

donde $\#y$ es el número natural representado por la cadena binaria y. Ejemplificando la

ecuación 1.1: para el caso de la cadena vacía se tiene:

$$x \rightarrow \#(1) - 1 = 0$$

y para la cadena 1010:

$$x \rightarrow \#(11010) - 1 = 25$$

En los ejemplos anteriores se puede observar que dada una entrada binaria se le concatena una unidad a la izquierda, la nueva cadena se transforma a base decimal y se le resta otra unidad, dando como resultado un número dentro del dominio de los números naturales.

Inversamente, es posible codificar cualquier clase de dato manejado por los formalismos anteriores como una cadena definida en $\{0, 1\}^*$. Bajo estas relaciones unívocas, todos ellos pueden simularse mutuamente y considerarse computacionalmente equivalentes [Koz97].

A pesar de ello, la noción de *efectivamente computable* seguía siendo informal ya que no podía cumplir uno de los requisitos que demanda el método: la no exigencia de conocimiento o ingenio. Uno de los logros de Turing en su ensayo de 1936 fue presentar un predicado formalmente exacto, con el cual el predicado informal: "es posible calcularlo por medio de un método eficaz"; podía ser reemplazado. Church hizo lo mismo. Ambos predicados resultaron ser equivalentes en el sentido de que cada uno escoge el mismo conjunto de funciones matemáticas.

El concepto formal propuesto por Turing es la *computabilidad por la Máquina de Turing*: siempre que haya un método eficaz para la obtención de los valores de una función matemática, la función puede ser calculada por una Máquina de Turing. Teóricamente, una Máquina de Turing puede calcular cualquier función normalmente computable, por ello, muchas veces se considera *computable* en el sentido de *computable por una Máquina de Turing* [Koz97].

Tesis de Turing:

LCMs(Logical Computing Machines: expresiones de Turing para las Máquinas de Turing) pueden hacer cualquier cosa que podría describirse como "regla general" o "puramente mecánico" [Cop08].

posteriormente, Turing añadió:

Esto es lo suficientemente bien establecido que se ha acordado entre los lógicos que "calculable por medio de una LCM" es la representación correcta de tales frases [Cop08].

Church y Kleene establecieron que la clase de funciones λ-definibles que propusieron era idéntica a la clase de funciones recursivas propuesta por Gödel y Herbrand, gracias a ello Turing estableció que su modelo era equivalente a la clase de funciones λ-definibles si éste se limitaba a funciones de números enteros positivos. Gracias a ello, Church enunció:

Tesis de Church:

Una función de enteros positivos es efectivamente calculable sólo si es recursiva [Cop08].

Finalmente, el término *"tesis de Church-Turing"* parece haber sido introducido por Kleene:

Así que las tesis de Turing y de Church son equivalentes. Deberíamos referirnos en general a ambas como la tesis de Church, o en relación con una de sus ... versiones que se refiere a "Máquinas de Turing" como la "tesis de Church-Turing" [Cop08].

La **tesis Church-Turing** se refiere a la noción de un *método "efectivo"* en lógica y matemáticas. Un método o procedimiento M para lograr algún resultado deseado es llamado *"efectivo"* ó *"mecánico"* sólo en caso de que:

1. *M se establezca en términos de un número finito de instrucciones exactas (cada instrucción expresada por medio de un número finito de símbolos),*

2. *M, si se lleva acabo sin errores, produzca el resultado deseado en un número finito de pasos,*

3. *M pueda, en práctica o en principio, llevarse acabo por un ser humano sin ayuda de ninguna maquinaria salvo lápiz y papel, y*

4. *M no exija conocimiento o ingenio por parte del ser humano para llevarse acabo [Cop08].*

1.4. Clase P

La clase P consiste en todos los problemas de decisión que pueden ser resueltos por una Máquina de Turing Determinista de una cinta (MTD) en tiempo polinómico. La MTD es representada esquemáticamente en la Figura 1.3. El tiempo empleado por una MTD, en el cálculo de un programa M, dada una entrada x, es el número de pasos que ocurren en ese cálculo hasta que un estado de paro es introducido [Gar79].

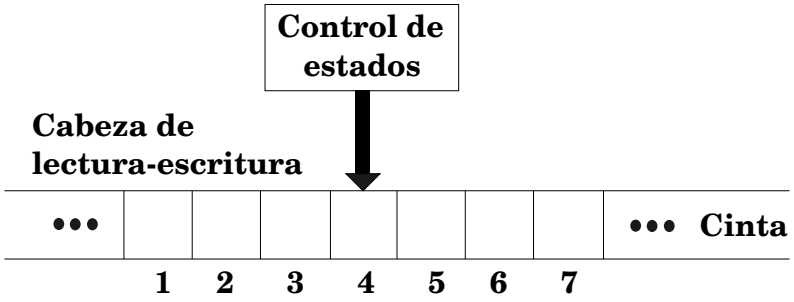

Figura 1.3: Máquina de Turing Determinista de una cinta (MTD). Consiste en un *control de estados*, una *cabeza de lectura-escritura* y una *cinta infinita* dividida en celdas, las cuales pueden contener cualquier símbolo definido dentro del alfabeto de la cinta [Gar79].

Definición 4. [Gar79] **La clase P**

$$P = \{L : \textit{existe un programa } M \textit{ para el cual } L = L_M\}$$

donde,

- M es un programa que puede ser ejecutado en tiempo polinómico por una MTD en el cual existe un polinomio p tal que $\forall n \in Z^+, T_M(n) \leq p(n)$; y
- L_M es el lenguaje reconocido por M definido como

$$L_M = \{x \in \Sigma^* : M \textit{ acepta } x\}$$

La clase P juega un papel principal en la complejidad computacional y es importante porque:

1. P es invariante para todos los modelos computacionales que son polinómicamente equivalentes a una MTD, y
2. P corresponde a la clase de problemas considerados como *tratables* [Sip13].

El punto 1. indica que P es una clase matemáticamente robusta, no es afectada por el modelo computacional que se esté utilizando y el punto 2. indica que P es relevante para el punto de vista práctico, es realizable, por lo tanto, se dice que los problemas en la clase P pueden ser eficientemente resueltos.

Algunos problemas pertenecientes a clase P son:

- **la ruta más corta entre dos vértices**: dado un grafo $G = (V, E)$ de longitud $l(e) \in Z^+$ para cada $e \in E$, vértices $a, b \in V$ y un entero positivo B ¿Existe un camino entre a y b en G de longitud total igual o menor a B?,

- **cubierta de aristas**: dado un grafo $G = (V, E)$ y un entero positivo K, ¿Existe un $E' \subseteq E$ con $|E'| \leq K$ tal que para cada $v \in V$ haya algún $e \in E'$ para el cual $v \in e$?, y
- **reducción transitiva**: dado un grafo dirigido $G = (V, A)$ y un entero positivo K, ¿Existe un $A' \subseteq V \times V$ con $|A'| \leq K$ tal que para toda $u, v \in V, G' = (V, A')$ contiene una ruta de u a v si y solo si G la contiene? [Gar79].

1.5. Clase NP

La clase NP consiste en todos los problemas de decisión que pueden ser resueltos por una Máquina de Turing No-Determinista de una cinta (MTND) en tiempo polinómico. La MTND es representada esquemáticamente y descrita en la Figura 1.4.

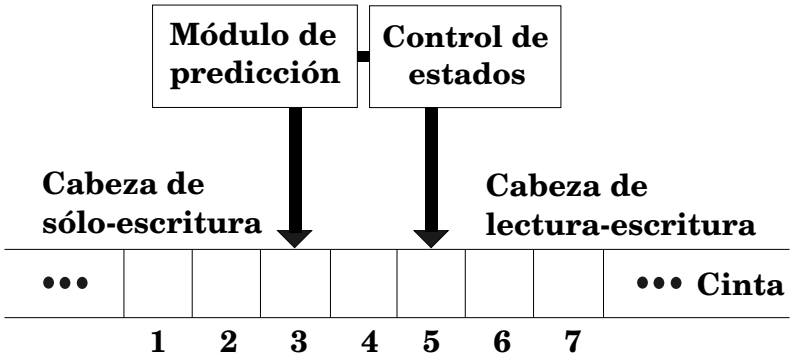

Figura 1.4: Máquina de Turing No-Determinista de una cinta (MTND). A diferencia de una MTD, este modelo incluye un *módulo de predicción* (con su propia cabeza de sólo-escritura) el cual sirve para suministrar una estimación de la respuesta para un problema deseado [Gar79].

Definición 5. [Gar79] **La clase NP**

$$NP = \{L : \textit{existe un programa } M \textit{ para el cual } L = L_M\}$$

donde,

- M es un programa que puede ser ejecutado en tiempo polinómico por una MTND en el cual existe un polinomio p tal que $\forall n \in Z^+, T_M(n) \leq p(n)$; y

- L_M es el lenguaje reconocido por M definido como

$$L_M = \{x \in \Sigma^* : M \ acepta \ x\}$$

Es importante notar que el concepto de "tiempo polinómico" en este contexto cobra otro sentido. Un *algoritmo no determinista*, compuesto por dos estados: uno de "suposición" y otro de "verificación", que resuelve un problema de decisión Π se dice que opera en "tiempo polinómico" ó "tiempo polinómico no determinista" si existe un *polinomio* p tal que, para cada instancia $I \in Y_\Pi$, hay una probable estructura S que conduce a un estado de verificación determinista que afirma a I y a S dentro del tiempo p [Gar79].

Un subconjunto de la clase NP son los problemas *NP-Completos*. La palabra "Completos" es usada para referir a los problemas más difíciles de la clase NP, los cuales son de mucho interés para las áreas de matemáticas, ciencias de la computación e investigación de operaciones ya que algunos problemas importantes son conocidos por ser *NP-Completos*. Algunos de ellos son:

- **conjunto independiente**: dado un grafo $G = (V, E)$ y un entero positivo $Kleq|V|$, ¿existe un conjunto independiente de vértices en G de tamaño igual o mayor a K?,
- **la ruta más larga entre dos vértices**: dado un grafo $G = (V, E)$ de longitud $l(e) \in Z^+$ para cada $e \in E$, vértices $a, b \in V$ y un entero positivo B ¿Existe un camino entre a y b en G de longitud total igual o mayor a B?, y
- ***clique***: dado un grafo $G = (V, E)$ y un entero positivo $K \leq |V|$, ¿existe un subconjunto de vértices donde cada dos de ellos se encuentren unidos por una arista y cuyo tamaño sea igual o mayor a K? [Gar79].

La teoría de la *NP-Completez* fue formulada para ser aplicada a este tipo de problemas.

Los principios de dicha teoría se establecieron en el artículo de Stephen A. Cook[13], presentado en 1971, titulado *The Complexity of Theorem Proving Procedures*. En él, menciona que:

- si se tiene una "reducción en tiempo polinómico" de un problema a otro, esto garantiza que cualquier *algoritmo de tiempo polinómico* para el segundo problema puede ser convertido a otro *algoritmo de tiempo polinómico* que corresponda al primer problema,
- muchos de los problemas aparentemente intratables que se encuentran en la práctica, cuando se enuncian como *problemas de decisión*, pertenecen a la clase NP,
- un problema particular en la clase NP, llamado "problema de satisfactibilidad" (SAT), tiene la propiedad de que cualquier otro problema de la clase NP puede ser reducido "polinómicamente" a él. Si éste problema puede ser resuelto por un *algoritmo de tiempo polinómico*, entonces también lo puede cualquier problema perteneciente a NP y si cualquier problema en NP es intratable, entonces el "problema de satisfactibilidad" también debe ser intratable. En este sentido, el "problema de satisfactibilidad" es el "más difícil" en la clase NP, y
- otros problemas en la clase NP podrían compartir con el "problema de satisfactibilidad" la propiedad de ser los "más difíciles" miembros de la clase NP [Coo71].

Más tarde Richard M. Karp[14] presentó en 1972 un artículo titulado *Reducibility Among Combinatorial Problems* donde demuestra que muchos problemas combinatorios conocidos, en versiones de problemas de decisión, son igual de "difíciles" que el "problema de satisfactibilidad" [Kar72]. Desde entonces, una amplia variedad de problemas se han demostrado equivalentes en dificultad a éstos últimos y por consiguiente, pertenecientes a la clase *NP-Completa* [Gar79].

[13] `http://www.cs.toronto.edu/~sacook/`
[14] `http://www.eecs.berkeley.edu/Faculty/Homepages/karp.html`

Finalmente, la meta del estudio de la teoría de la *NP-Completez* es la identificación de problemas pertenecientes a la clase P que impliquen la igualdad entre las clases P y NP, para ello se debe cumplir que:

Teorema 1. [Kar72] Si $L \in NP$ entonces $L \propto SAT$ [15]

Corolario 1. [Kar72] $P = NP \iff SAT \in P$

Si $SAT \in P$, entonces, por cada $L \in NP,\ L \in P$, dado que $L \propto SAT$. Si $SAT \notin P$ entonces $SAT \in NP,\ P \neq NP$ [Kar72].

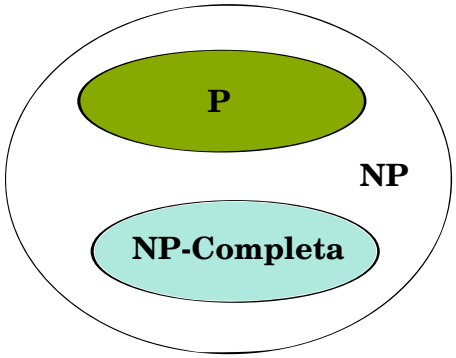

Figura 1.5: Relación entre las clases P, NP y NP-Completa. La imagen muestra a las clases P y NP-Completa como conjuntos disjuntos contenidos en la clase NP [Cor09].

En este capítulo introductorio se presentaron los conceptos fundamentales que servirán para analizar y modelar el problema de estudio referente a la asignación de grupos dentro de la FI. Los siguientes capítulos hablarán sobre la problemática en cuestión, la herramienta propuesta para ofrecerle soluciones y los resultados obtenidos con ella al evaluar algunos casos particulares.

[15] El símbolo $\propto$ denota "es reducible a".

Capítulo 2

El problema de administración de grupos dentro de la Facultad de Ingeniería

2.1. Descripción general

La Facultad de Ingeniería se encuentra ubicada en el Circuito Escolar en Ciudad Universitaria[16], cuenta con 23 edificios que albergan laboratorios, bibliotecas, auditorios y demás aulas utilizadas para coordinar y realizar las diversas actividades académicas relacionadas con la facultad. Dentro de ella, las divisiones académicas (Cuadro 2.1) se encargan de organizar e impartir las diferentes carreras (Cuadro 2.2) las cuales están acreditadas por el Consejo de Acreditación de la Enseñanza de la Ingeniería (CACEI).[17]

Según el *Informe de actividades 2010* expuesto por el director de la Facultad, la población escolar del semestre 2011-1 fue de 12,089 estudiantes de licenciatura distribuidos en 4,944

[16]Avenida Universidad No. 3000, Universidad Nacional Autónoma de México, C.U., Delegación Coyoacán, Distrito Federal, C.P. 04510.

[17]`http://www.ingenieria.unam.mx`

grupos con 566 asignaturas de teoría y laboratorio [Gue11], las cifras son motivadoras por el incremento de universitarios en formación, pero tal incremento demanda recursos que las autoridades se deben ocupar en responder, como es el caso de ofrecer grupos suficientes que cubran la demanda generada por los alumnos inscritos.

En la FI se presenta el problema de la administración de grupos previo al proceso de inscripción, por ello las asignaciones generadas a veces no son tan favorables para el alumno e inclusive para el profesor, y esto se debe a que el método actual no toma en cuenta algunas consideraciones y preferencias, las cuales ayudarían a mejorar la distribución de grupos. Esta problemática puede modelarse como un problema de *timetabling*,[18] el cual se describe en la siguiente sección.

Divisiones académicas de la Facultad de Ingeniería
División de Ciencias Básicas
División de Ingeniería Mecánica e Industrial
División de Ingeniería Civil y Geomática
División de Ingeniería Eléctrica
División de Ingenierías de Ciencias de la Tierra
División de Ciencias Sociales y Humanidades
División de Educación Continua y a Distancia[19]

Cuadro 2.1: Divisiones académicas de la Facultad de Ingeniería.

Carreras impartidas dentro de la Facultad de Ingeniería
Ingeniería Civil
Ingeniería Geomática
Ingeniería Geofísica
Ingeniería Geológica
Ingeniería de Minas y Metalurgia
Ingeniería Petrolera
Ingeniería Eléctrica y Electrónica
Ingeniería en Computación
Ingeniería en Telecomunicaciones
Ingeniería Mecánica
Ingeniería Industrial
Ingeniería Mecatrónica

Cuadro 2.2: Carreras impartidas dentro de la Facultad de Ingeniería.

[18] El término *timetabling* puede traducirse como *programación de horarios* o *calendarización*.

[19] Su sede se encuentra ubicada en el Palacio de Minería: Calle de Tacuba No. 5, Centro Histórico, Delegación Cuauhtémoc, Distrito Federal.

2.2. Timetabling

El *timetabling* es un problema combinatorio multi-restricción *NP-Completo* para el cual no se conoce ningún algoritmo que pueda resolverlo en *tiempo polinómico* [Per06].

Definición 6. [Eve76] **El problema de programación de horarios es NP-Completo.** El problema de programación de horarios (*timetable problem, TT*) es un modelo matemático sobre la programación del plan de enseñanza en una escuela. De hecho, es un modelo bastante ingenuo ya que hace caso omiso de varios factores que sin duda juegan un papel en la práctica. Sin embargo, incluso una restricción adicional sigue dando lugar a un problema NP-Completo.

Definición (TT). Dado los siguientes datos:

1. un conjunto finito H(de horas en la semana);

2. una colección $\{T_1, T_2, \ldots, T_n\}$, donde $T_i \subseteq H$; (hay n profesores y T_i es el conjunto de horas durante las cuales el $i-ésimo$ profesor esta disponible para dar clase);

3. una colección $\{C_1, C_2, \ldots, C_m\}$, donde $C_j \subseteq H$; (hay m clases y C_j es el conjunto de horas durante las cuales la $j-ésima$ clase esta disponible);

4. una matriz R de orden $n \times m$ formada por enteros no negativos; (R_{ij} es el número de horas en las cuales el $i-ésimo$ profesor es requerido para impartir la $j-ésima$ clase).

El problema es determinar si existe una función de satisfacción

$$f(i,j,h) : \{1,\ldots,n\} \times \{1,\ldots,m\} \times H \to \{0,1\} \tag{2.1}$$

(donde $f(i,j,h) = 1$ si y solo si el profesor i imparte la clase j durante la hora h) tal que:

(a) $f(i,j,k) = 1 \Rightarrow h \in T_i \cap C_j$;

(b) $\sum_{h \in H} f(i,j,h) = R_{ij}$ para toda $1 \leq i \leq n$ y $1 \leq j \leq m$;

(c) $\sum_{i=1}^{n} f(i,j,h) \leq 1$ para toda $1 \leq j \leq m$ y $h \in H$;

(d) $\sum_{j=1}^{m} f(i,j,h) \leq 1$ para toda $1 \leq i \leq n$ y $h \in H$.

(a) asegura que un evento se lleva a cabo sólo si tanto el profesor como la clase están disponibles; (b) asegura que el número de reuniones durante la semana entre el profesor i y la clase j son las requeridas por el número R_{ij}; (c) asegura que ninguna clase tiene más de un profesor al mismo tiempo; y (d) asegura que ningún profesor esta impartiendo dos clases simultáneamente.

En otras palabras, el *timetabling* se puede definir como la asignación de un conjunto de elementos (por ejemplo, clases, aulas, profesores) dentro de un número limitado de periodos de tiempo sujeto a un conjunto de restricciones que se buscan cumplir de la mejor manera posible, éstas son clasificadas generalmente en dos tipos: fuertes y débiles. Las **restricciones fuertes** son condiciones que bajo ninguna circunstancia deben ser infringidas y son utilizadas para encontrar soluciones realizables; las **restricciones débiles** son condiciones deseables de cumplir, pero dicha exigencia puede ser omitida si es necesario y generalmente son utilizadas para entregar un costo a la función objetivo, la cual será optimizada.

Las diversas restricciones hacen del *timetabling* un problema de optimización multi-objetivo, donde no es posible utilizar métodos de búsqueda exhaustiva ya que el tiempo y costo computacional no son viables por ser sumamente elevados y desde el punto de vista práctico inalcanzables.

Para los problemas de *timetabling*, se pueden definir dos tipos de soluciones: la **solución válida** y la **solución factible**. La primera es aquella que cumple con todas las restricciones fuertes especificadas dentro del dominio del problema, dejando algunos eventos sin asignar;

por su parte, la segunda, de igual manera, respeta las restricciones fuertes y además asigna todos los eventos [Lew07]. Para ambos casos, entre más restricciones débiles se cumplan, mejor será la solución. La Figura 2.1 ilustra ambos tipos de soluciones.

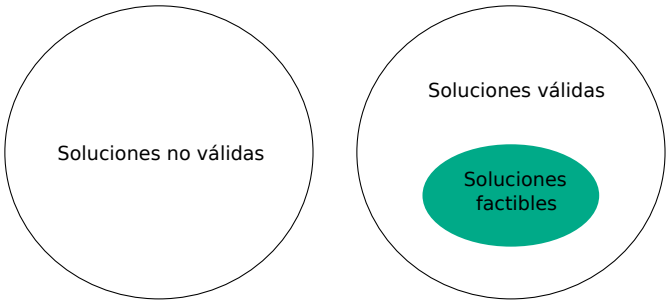

Figura 2.1: Tipos de soluciones para el *timetabling*. El conjunto de *Soluciones no válidas* agrupa a todas aquellas soluciones que rompen al menos una restricción fuerte. Por otro lado, el conjunto de *Soluciones válidas* esta formado por soluciones que cumplen todas las restricciones fuertes especificadas. Dentro de este conjunto, cuando alguna solución tiene la totalidad de eventos asignados, la solución pasa a formar parte del subconjunto de *Soluciones factibles*.

Diferentes variantes del problema de *timetabling* han sido propuestas en la literatura, las cuales se diferencian entre si de acuerdo a el tipo de restricciones y objetivos involucrados, algunas de ellas son:

- **calendarización basada en el plan de estudios**: Consiste en la programación semanal de varios cursos universitarios dentro de un determinado número de aulas y periodos de tiempo, donde los conflictos entre los cursos se establecen de acuerdo a los planes de estudio publicados por la universidad [Gas07],
- **calendarización de exámenes**: Se define como el problema de asignación de horarios para la aplicación de exámenes en un número limitado de periodos de tiempo [Pet04], y
- **calendarización basada en la inscripción de cursos**: En esta variante, los estudiantes tienen la opción de elegir los cursos que desean atender, y la programación de horarios se realiza posterior a este proceso utilizando la información generada [Lew07].

El *timetabling* es un problema de interés e importancia para diversas áreas, por ello se continúa investigando acerca de algoritmos que puedan mejorar los actualmente existentes, muchas veces combinando algunos de ellos (aproximaciones tipo portafolio o meta-algoritmos). Por ejemplo, las conferencias Practice and Theory of Automated Timetabling (PATAT)[20] son eventos internacionales dedicados a publicar sobre avances en el diseño, uso, pruebas y comparaciones entre algoritmos aplicados a diferentes problemáticas; por otro lado, la International Timetabling Competition (ITC) [21] es una competencia internacional donde se ponen aprueba algoritmos diseñados por distintos estudiantes e investigadores para ofrecer soluciones a diferentes problemas de *timetabling* especificados.

2.3. Análisis del problema

Actualmente las aulas con los que cuenta la FI se encuentran repartidas entre las divisiones académicas y la asignación de grupos se realiza básicamente de la siguiente manera[22]:

1. Para cada uno de los departamentos de cada división, los jefes de departamento[23] ofertan los grupos "necesarios" [24] para cubrir la demanda de la próxima inscripción.

2. Los jefes de departamento se encargan de buscar a los profesores que cubran la demanda de grupos propuestos, contratando nuevo personal de ser necesario y pidiendo la confirmación de los profesores respecto a su horario de acuerdo a la asignación del semestre anterior.

3. Los secretarios académicos revisan las propuestas de los jefes de departamento correspondientes a su división, para verificar que se encuentren en regla. En caso de que

[20] `http://www.patat2012.com/`
[21] `http://www.utwente.nl/ctit/hstt/itc2011/welcome/`
[22] Consideraciones tomadas hasta la fecha de elaboración de esta tesis.
[23] Para la División de Ciencias Básicas, el actor es el coordinador de cada materia.
[24] En realidad, los grupos que llaman "necesarios" son inicialmente los mismos que se utilizaron el semestre anterior.

una propuesta presente alguna irregularidad se hacen las respectivas observaciones y se rechaza, en caso contrario, se acepta.

4. Para el proceso de inscripción, se publican los horarios de los grupos con aulas establecidas, con excepción de algunos de ellos. [25]

5. Los secretarios académicos resuelven conflictos (las excepciones del punto anterior) y presentan las asignaciones poco antes del inicio de semestre.

6. Una vez iniciado el semestre, durante la primera semana se realiza un proceso que permite al alumno de manera opcional modificar su inscripción, para ese entonces, los horarios publicados pudieron haber sido modificados de acuerdo a la demanda presentada. Posterior a este proceso, se presentan las asignaciones finales.

Este procedimiento puede llevar algunas semanas, la mayoría de las acciones se basan en asignaciones anteriores, los conflictos primero se tratan de resolver internamente en cada división académica y en caso de ser necesario se pide a otras divisiones la autorización para asignar grupos en aulas que se encuentran bajo su "pertenencia". Además, se pueden presentar cambios de última hora posteriores a la publicación de horarios, y en algunos casos las modificaciones sólo se realizan en la página de internet y no en los horarios pegados en las puertas de las aulas.

2.4. Formulación matemática del problema

El problema de asignación de grupos que enfrenta la Facultad de Ingeniería, descartando los casos especiales que requieren asignarse manualmente, puede modelarse de la siguiente manera:

[25] Otra excepción es la falta de profesor para el grupo, presentada como *Profesor por asignar*.

Definición 7. *Dada una 12-tupla:*

$$\Phi = (T, P, S, A, G, f_p, f_s, f_m, f_n, q, R_f, R_d)$$

donde T, P, S, A, G, R_f, R_d *son todos conjuntos finitos y*

1. T *es un conjunto de "ventanas de tiempo"*[26]
2. P *es un conjunto de profesores,*
3. S *es un conjunto de aulas,*
4. A *es un conjunto de asignaturas,*
5. G *es un conjunto de grupos, donde* $G = \{\ g \mid g = (p, a), p \in P, a \in A\}$,
6. $f_p : P \to 2^T$ *es la disponibilidad del profesor,*
7. $f_s : A \to 2^S \setminus \{\}$, *es el conjunto de aulas donde puede impartirse una asignatura,*
8. $f_m : A \to \mathbb{N}^+$, *son los minutos que dura a la semana una asignatura,*
9. $f_n : A \to 2^{\{1,2,3\}} \setminus \{\}$, *es el número de sesiones en las que puede impartirse una asignatura,*
10. q *es la resolución o duración en minutos de una "ventana de tiempo".*

El problema de asignación de grupos que enfrenta la Facultad de Ingeniería consiste en encontrar un conjunto de asignaciones Θ^*, *donde*

$$\Theta^* = \max_{\Theta} |\Theta| \tag{2.2}$$

[26]Las *ventanas de tiempo* son intervalos regulares en los que se dividen las disponibilidades.

y

$$\Theta = \{\ \theta \mid \theta = (g, h, s),\ g = (p, a),\ g \in G,\ h \in f_p(h),\ s \in f_s(a)\ \}$$

Sujeto a

11. Un conjunto de restricciones fuertes R_f, descritas como:

11.1 Sólo se permite una asignación por grupo

$$\forall i, j\ (\theta_i, \theta_j \in \Theta \wedge i \neq j) \Rightarrow g_i \neq g_j \tag{2.3}$$

11.2 Ningún profesor debe quedar obligado a asistir a más de un evento al mismo tiempo. Esto es, dadas las asignaciones

$$\theta_i = (g_i, h_i, s_i),\ g_i = (p_i, a_i),\ \theta_i \in \Theta$$

Se cumple que

$$\forall i, j\ (p_i = p_j \wedge i \neq j) \Rightarrow h_i \cap h_j = \{\} \tag{2.4}$$

11.3 En cada caso, el aula debe ser lo suficientemente grande para todos los estudiantes que asistan a la sesión[27] *y de satisfacer todas las características requeridas por el evento.*[28]

11.4 Sólo un evento tiene lugar en cada aula en cualquier intervalo de tiempo. Esto es, dadas las asignaciones θ_i se cumple que

$$\forall (s_i = s_j \wedge i \neq j) \Rightarrow h_i \cap h_j = \{\} \tag{2.5}$$

[27] En la práctica, no siempre se toma en cuenta esta restricción debido a que los administrativos no cuentan con el valor exacto de la capacidad de todas las aulas.

[28] Las aulas válidas para cada asignatura son especificadas en la entrada, por lo que esta restricción es manejada previamente y definida por los administrativos.

11.5 Los eventos sólo se deben de asignar a intervalos de tiempo que son predefinidos como "disponibles" para dichos eventos.[29]

12. Un conjunto de restricciones débiles R_d, donde $R_d = \{\}$.

La **Definición 6** busca encontrar, si existe, la función de satisfacción (2.1), mientras que la **Definición 7** busca maximizar la cardinalidad de Θ, representada por las barras verticales en la fórmula 2.2. Por lo tanto, el problema descrito de la FI es un problema de optimización.

2.5. Técnicas de solución para el timetabling

Con el paso de los años se han desarrollado algoritmos que buscan ofrecer soluciones viables para los problemas de restricción multi-objetivo, debido a que no existe un método que siempre garantice encontrar una *solución factible*. Las técnicas o algoritmos tradicionales no funcionan en problemas de esta naturaleza ya que no toman en cuenta los conflictos que causan entre si las restricciones, las cuales incrementan la dificultad del problema, además de que el tiempo de búsqueda de una solución sería excesivo e incluso intratable.

En el área de Inteligencia Artificial, para el diseño de algunos algoritmos se han usado como base teórica a la física, biología, sociología, psicología, ecología e inclusive la música. La analogía que se hace entre los algoritmos y algunos conceptos de ciertas disciplinas, ha permitido generar metodologías que ofrecen soluciones adecuadas (un buen desempeño experimental en tiempo y calidad de respuesta) a diversas problemáticas, muchas veces se buscan mejorar los procedimientos conocidos. Algunos ejemplos pueden ser los *Algoritmos Genéticos* basados en el proceso de selección natural [Mit98], el algoritmo de *Harmony Search* el cual utiliza las experiencias de los músicos en la improvisación jazzista [Gem10], o el algoritmo de *Ant Colony Optimization* inspirado por el comportamiento en la búsqueda de alimento de las colonias de hormigas [Dor04].

[29] En la siguiente sección, se explicará que las asignaciones son construidas siempre bajo este criterio.

Los algoritmos que se utilizan para resolver los problemas de *timetabling* y otros problemas similares de complejidad NP-Completa [30] se pueden clasificar en dos grupos: **algoritmos de búsqueda local** y **algoritmos basados en poblaciones**.[31]

Los **algoritmos de búsqueda local** son aquellos que parten de una solución inicial generada aleatoriamente o hallada con otro algoritmo y a esa solución se le aplica una serie de transformaciones de algún conjunto dado para mejorarla (Cuadro 2.3). Un conjunto más específico dentro de este tipo de algoritmos se encuentran las técnicas de resolución para problemas de satisfacción de restricciones, *Constraint Satisfaction Problems (CSP)*, que introducen propagación de restricciones en los algoritmos de búsqueda y por ello, generalmente son completos [32] (Cuadro 2.4).

Los **algoritmos basados en poblaciones**, en lugar de iniciar con una solución, parten de un conjunto de soluciones para aprovechar la exploración paralela del espacio. En estos algoritmos no se trata simplemente de correr múltiples búsquedas locales de manera simultánea, sino de combinar de alguna manera la información conjunta de dos o más soluciones para generar nuevos resultados que reemplazan soluciones de la población original. (Cuadro 2.5)

Por lo general, los algoritmos utilizan diferentes criterios para seleccionar cuál entre varios cursos de acción tomar para lograr algún objetivo de la manera más eficaz posible. Estos criterios, conocidos como *heurísticas*, se buscan que sean simples y que puedan discriminar correctamente entre las posibles decisiones a elegir [Pea84]. En un orden superior, un **metaheurístico** se refiere también a un criterio, pero ésta vez, que guía y modifica a otros para producir soluciones que superen a las generadas usando solamente *heurísticas* [Glo97].

Entre los distintos métodos que se han estudiado y demostrado que son aceptables

[30] En la sección *1.5. Clase NP* se mencionan algunos ejemplos.

[31] Esta clasificación no contempla todas las técnicas utilizadas para este tipo de problemas, entre ellas se encuentran Sistemas Basados en Conocimiento, algoritmos de Aprendizaje Máquina y Sistemas Expertos.

[32] Una algoritmo es considerado *completo* cuando siempre regresa una solución y ésta no necesariamente es correcta.

para ofrecer soluciones a los problemas de *timetabling* se encuentran: *Tabu Search (TS)* [Gas01] [Whi01][Sou04], *Simulated Annealing (SA)* [Tho96][Fra07][Qu,09], *Non-dominated Sorting Genetic Algorithm-II (NSGA-II)* [Dat06][Jat11], *Harmony Search (HS)* [Al-09][Al-10] y *Squeaky Wheel Optimization (SWO)* [Bur04] [Fen05]. La selección en específico de estos algoritmos se debe a distintas razones: el *Tabu Search* y el *Simulated Annealing* son meta-heurísticos que han sido reconocidos por su eficacia para resolver problemas de asignación de horarios, el primero es un algoritmo que usa una búsqueda agresiva y memoria, mientras que en el segundo las soluciones generadas cambian de estado de acuerdo a una función de probabilidad especificada; *Non-dominated Sorting Genetic Algorithm-II* y *Harmony Search* utilizan el concepto de no-dominancia en la frontera Pareto, que en problemas con restricciones multi-objetivo, resulta de mucha utilidad para decidir entre un conjunto de posibles soluciones cuál de ellas es la más adecuada; y finalmente *Squeaky Wheel Optimization* y el funcionamiento cíclico de sus bloques constructor/analizador/asignador-de-prioridades puede ser una buena herramienta adaptativa a las restricciones específicas del problema.

Puesto que en la literatura se encuentran investigaciones acerca de la aplicación de todos ellos a problemas similares, se puede decir, que todos son buenos candidatos. Inclusive se pueden hacer modificaciones o combinaciones con los algoritmos mencionados para crear algoritmos híbridos o hiper-heurísticos que se ajusten mejor a las condiciones requeridas. En las sesiones siguientes se explicarán más a detalle los algoritmos seleccionados.

Algoritmos de búsqueda local
Simulated Annealing
Tabu Search
Squeaky Wheel Optimization
Walk Down Jump Up

Cuadro 2.3: Ejemplos de algoritmos de búsqueda local.

Técnicas para problemas de satisfacción de restricciones
Maintaining Arc Consistency
Generate and Test
Backtraking
Iterative Forward Search

Cuadro 2.4: Ejemplos de algoritmos para problemas de satisfacción de restricciones.

Algoritmos basados en poblaciones
Particle Swarm Optimization
Harmony Search
Non-dominated Sorting Genetic Algorithm-II
Ant Colony Optimization

Cuadro 2.5: Ejemplos de algoritmos basados en poblaciones.

2.6. Algoritmos de búsqueda local

2.6.1. Tabu Search (TS)

Es un meta-heurístico basado en la idea de que un movimiento informado es más útil que uno al azar debido a la información que éste aporta, aceptando siempre el movimiento siguiente a pesar de que no sea el mejor. TS utiliza búsqueda agresiva, es decir, evita que la búsqueda quede atrapada en un óptimo local que no sea global; además posee memoria adaptativa (memoria selectiva) y exploración sensible (busca buenas características de las soluciones). El algoritmo comienza en una solución cualquiera elegida al azar y en cada iteración selecciona la mejor solución del vecindario. Para no volver recurrente la búsqueda, se utiliza una memoria dinámica a corto plazo que lista las soluciones más recientemente visitadas: la lista tabú. Esta memoria puede ser explícita (memoriza toda la solución encontrada) o atributiva (memoriza solo las características más importantes de la solución), esta última reduce el riesgo de hacer procesos cíclicos pero puede prohibir la visita a soluciones no exploradas que podrían resultar interesantes para la búsqueda. También se puede utilizar una memoria a largo plazo la cual almacena las frecuencias de las ocurrencias de atributos de las soluciones visitadas, aunque ésta se restringe al caso especial donde las soluciones tabú se mueven al peor de los casos [Glo97]. El uso del tipo de memoria y las técnicas de penali-

zación o premiación dependerá de la habilidad del programador para adaptar el algoritmo a un problema en específico.

2.6.2. Simulated Annealing (SA)

Es un método basado en el recocido de los metales, es decir, imita la manera en que los sistemas termodinámicos pasan de un nivel de energía a otro. Si un metal es enfriado lentamente, éste formará una pieza lisa porque sus moléculas han construido una estructura cristalina. Esta estructura cristalina representa el estado de energía mínima, o la solución óptima, para un problema de optimización. De lo contrario, si un metal es enfriado muy rápido, éste formará una pieza áspera e irregular cuyos bordes generados representan los mínimos y máximos locales del problema.

Primeramente se fijan los parámetros de temperatura inicial y temperatura final, después el SA comienza eligiendo aleatoriamente alguna configuración que resuelve el problema. Posteriormente, genera n configuraciones nuevas, una a la vez y siempre derivada de la configuración anterior, para encontrar el equilibrio térmico. En cada movimiento, el sistema tiene cierta probabilidad de cambiar su configuración a una peor. La distribución de probabilidad con que el sistema acepta una configuración de mayor energía se puede definir como $e^{-\frac{E_2-E_1}{kT}}$, donde E_1 es el costo de la configuración actual, E_2 es el costo de la configuración modificada y k, la constante de Boltzmann,[33] en este caso, es alguna constante elegida para adaptarse a un problema específico, aunque es típico no considerarla o igualarla a uno. Si una configuración nueva tiene un costo menor, y por lo tanto es mejor que la configuración actual, entonces será aceptada automáticamente, en el caso contrario, una variable aleatoria con la distribución de probabilidad asociada, determina cuándo la solución es aceptada o rechazada.

[33]La constante de Boltzmann es una constante física equivalente a $1.3806488 \times 10^{-23} JK^{-1}$ que relaciona la energía a nivel individual de una partícula con la temperatura observada a nivel colectivo o mayor y es definida por el cociente de la constante de los gases R sobre la constante de Avogadro N_A [Bol13].

La temperatura cambia después de que este proceso de encontrar una configuración nueva, compararla con la configuración actual, y aceptarla o rechazarla, es realizado n veces. Al disminuir la temperatura, la probabilidad de seleccionar una configuración nueva es menor. Para terminar, este proceso se repite hasta que la temperatura de paro es alcanzada o hasta encontrar una solución aceptable [Sch11].

2.6.3. Squeaky Wheel Optimization (SWO)

Es un meta-heurístico usado para construir soluciones a partir de elecciones localmente óptimas, las cuales son analizadas para encontrar y tratar de resolver los puntos conflictivos que ayudan a mejorar la puntuación de la función objetivo. SWO trabaja mediante un ciclo formado por los bloques *Constructor*, el cual genera una solución local a partir de un ordenamiento dado (inicialmente puede ser aleatorio) usando un algoritmo glotón o de explotación sin *backtrack* que puede romper las restricciones fuertes del problema; *Analizador*, quien asigna un factor de penalización a los elementos del problema de acuerdo a su desempeño, llamado *blame factor*; y *Asignador de Prioridades* el cual utiliza dicho factor de penalización para modificar la secuencia previa de los elementos del problema y generar una nueva la cual servirá de entrada para el bloque *Constructor*. Este ciclo continua hasta alcanzar un límite especificado o una solución satisfactoria.

Este algoritmo opera en dos espacios de búsqueda: el de *prioridades* y el de *soluciones*. El *Constructor*, quien trabaja con secuencias ordenadas de elementos de acuerdo a sus prioridades, permite cambiar del espacio de prioridades al espacio de soluciones cuando genera alguna solución; en cambio, el *Analizador* y el *Asignador de Prioridades*, a partir de las soluciones generadas, cambian del espacio de soluciones al de prioridades cuando analizan los puntos conflictivos y ordenan la secuencia previa basándose en el *blame factor* para crear una nueva e iterar en el ciclo.

Metafóricamente, el funcionamiento de SWO asemeja el movimiento de una rueda que

rechina: se tiene una rueda a la cual se le ejerce una fuerza para ponerla en movimiento, por ello, se desplaza por una trayectoria fija y en una sola dirección hacia una meta. Durante ese movimiento, la rueda puede encontrar obstáculos en su camino, quienes la pueden estancar momentáneamente. Sin embargo, la rueda nunca deja de girar y al tratar de continuar su camino, pasa por encima de los obstáculos y produce un rechinido al hacerlo. Finalmente, la rueda llega a la meta. En este caso, la rueda representa un problema de optimización, la meta es una solución para dicho problema y el movimiento es el funcionamiento de SWO, un ciclo que resuelve los conflictos encontrados, los obstáculos del camino, y permite llegar a una solución. El movimiento en una sola dirección, representa al algoritmo usado por el bloque *Constructor* caracterizado por no utilizar *backtrack*.

2.7. Algoritmos de búsqueda basados en poblaciones

2.7.1. Non-dominated Sorting Genetic Algorithm-II (NSGA-II)

El NSGA-II es un algoritmo evolutivo elitista basado en poblaciones para problemas de restricciones multi-objetivo. En este método la población descendiente es creada en primera instancia usando la población de padres. Después de esto, las dos poblaciones son combinadas mediante un ordenamiento basado en el principio de no-dominancia: Sean v_1 y v_2 dos soluciones Pareto-óptimas donde v_1 domina a v_2 si:

1. v_1 no es peor que v_2 en ninguno de los objetivos, y

2. v_1 es estrictamente mejor que v_2 en por lo menos un objetivo.

De lo contrario v_1 y v_2 no son comparables [Deb00]. Posteriormente se clasifica en diferentes *frentes Pareto* y aunque esto requiere un mayor esfuerzo, se justifica por el hecho de permitir una verificación global de dominancia entre la población de padres y descendientes.

Una vez que el proceso de ordenamiento no-dominado ha finalizado, la nueva población es generada a partir de las configuraciones de los frentes no-dominados. Aquellos frentes que no pueden ser acomodados desaparecen. La idea es que siempre se promuevan las configuraciones que aseguren diversidad dentro del mismo frente Pareto [Wei09]. Cuando la población en su totalidad converge al frente Pareto óptimo, el algoritmo promueve que las soluciones estén distanciadas una de otra, para ello, se define un criterio de comparación basado en el estratificación de la población en fronteras Pareto y en la diversidad de las soluciones, dicho criterio se utiliza para decidir que individuo gana un torneo binario, que es el método utilizado para la selección de los individuos a reproducir, mientras que el cruzamiento y la mutación son utilizados para crear la población de descendientes.

2.7.2. Harmony Search (HS)

Es un método estocástico, basado en la experiencia adquirida por los músicos en la improvisación del jazz, que asigna una probabilidad de selección a cada valor de una variable de decisión, sea ésta continua o discreta, la cual se actualiza en cada iteración hasta encontrar el vector óptimo que representa la solución del problema. Este algoritmo usa la formulación de vectores para moverse hacia posibles soluciones y al igual que el NSGA-II, utiliza el criterio de la Frontera Pareto.

La estructura básica de HS es la siguiente:

1. formulación del problema (optimizar un problema sujeto a restricciones),

2. configuración de los parámetros del algoritmo (tamaño de la memoria armónica, consideración del rango de la memoria armónica, rango de ajuste de tono, máxima improvisación y ancho del traste),

3. afinación aleatoria para la inicialización de la memoria,

4. improvisación armónica (selección aleatoria, consideración de la memoria y ajuste de tono),

5. actualización de la memoria,

6. finalización de la improvisación (en caso de cumplir las condiciones iniciales, de lo contrario repite) y

7. cadencia (regresa la mejor armonía encontrada) [Gem10].

2.8. Algoritmos híbridos e hiper-heurísticos

Los algoritmos híbridos son combinaciones entre dos o más algoritmos cuyo objetivo es compensar o mejorar los algoritmos originales, entre ellos se encuentran los *hiper-heurísticos*, los cuales son una emergente metodología en búsqueda y optimización de "heurísticos que seleccionan heurísticos" [Bur09], es decir, dependiendo del problema el algoritmo escoge al mejor heurístico entre la gama que tiene adjunta para llegar a una solución, estos pueden ser "heurísticos de bajo nivel" (funciones) o "meta-heurísticos" (algoritmos).

Algunos *hiper-heurísticos* trabajan con reglas que se han extraído del aprendizaje y la adquisición de conocimientos. Estas extracciones pueden ser realizadas por ellos mismos o por otros algoritmos, por ello, en ocasiones son clasificados como Sistemas Expertos [Gar11].

Este capítulo, entre otras cosas, describió a grandes rasgos los algoritmos candidatos para ofrecer soluciones al problema presentado de la FI. Describirlos detalladamente queda fuera del alcance de este trabajo. En el próximo capítulo se retomarán dos de los algoritmos mencionados y se describirán las modificaciones realizadas.

Capítulo 3

SAG-FI: Sistema de Administración de Grupos para la Facultad de Ingeniería

3.1. Antecedentes

En septiembre de 2007, un grupo de estudiantes de la carrera de Ingeniería en Computación y profesores de la misma, comenzaron dentro del Laboratorio de Investigación para el Desarrollo Académico (LINDA) un proyecto que nombraron Sistema de Administración de Grupos para la Facultad de Ingeniería (SAG-FI) cuyo objetivo principal consistía en desarrollar un sistema automático para la administración de grupos dentro de dicha facultad, un proyecto con finalidad similar a la de otros tantos presentados anteriormente por diferentes alumnos y profesores. [34]

Posteriormente, la idea del proyecto fue presentada al Secretario Académico de la División de Ingeniería Eléctrica (DIE), el M.I. Aurelio Adolfo Millán Nájera, y a uno de los encargados

[34]Entrevista con el secretario de servicios académicos de la FI, Lic. Miguel Figueroa Bustos, noviembre 2011.

del proceso de inscripción de alumnos dentro de la facultad, el Ing. Jorge Ontiveros Junco, con la intención de promover la participación e interés de las autoridades para brindar información y apoyo en caso de ser necesario. El proyecto se trabajó y se dejaron algunos documentos y código como referencias para trabajo futuro.

En febrero de 2011, nuevamente un grupo de alumnos retomamos el proyecto, teniendo en cuenta las metas especificadas, planteamos nuevas ideas y líneas de acción para trabajar y cumplir el objetivo definido, entre ellas destacan las siguientes:

- desarrollar un algoritmo que asigne grupos de manera automática y que funcione como un *web service* para permitir la interacción con diferentes plataformas,
- desarrollar un sistema web que facilite la captura y validación de los datos necesarios para utilizar el algoritmo, e
- implementar una base de datos que permita almacenar la información ingresada por el sistema web y generada por el algoritmo.

Cuando platicábamos sobre las tecnologías que podríamos utilizar, decidimos probar con un enfoque no tradicional dentro del desarrollo de sistemas en la FI y seleccionamos el lenguaje de programación Erlang, el cual es un lenguaje funcional, concurrente y tolerante a fallos empleado en telecomunicaciones, comercio electrónico, telefonía por computadora, entre otras áreas[35], con ello nos permitiría desarrollar una infraestructura escalable y sólida pensada para atender múltiples peticiones simultáneas. Siguiendo con el uso de esta tecnología, se optó por utilizar una base de datos no relacional escrita en Erlang y de código abierto: CouchDB[36], la cual almacena la información en documentos con formato JavaScript Object Notation (JSON) [37] y permite realizar consultas tanto en Erlang como en JavaScript.

[35] `https://www.erlang.org`
[36] `https://couchdb.apache.org/`
[37] `http://www.json.org`

También se decidió programar el algoritmo en Erlang para utilizar operaciones concurrentes que permitieran manejar numerosas operaciones, algunas independientes de otras, afectando de manera positiva el tiempo de ejecución al operar sobre múltiples procesadores de manera simultánea.[38] A su vez, el algoritmo debía ser independiente del sistema web, con ello permitiría operarlo en un equipo remoto o mediante otros sistemas, por lo que era necesario establecer un formato ligero de entrada/salida y de fácil lectura. Convencidos de que el formato JSON utilizado por CouchDB cumplía esas características, decidimos usarlo en los archivos de entrada y de salida del algoritmo.

Finalmente el equipo se redujo a una sola persona, por lo que fue necesario delimitar el alcance de este trabajo, el cual se enfoca en uno de los puntos fundamentales para el desarrollo del SAG-FI: diseñar un algoritmo que ayude a la asignación automática de grupos.

3.2. Objetivo

Mi objetivo es desarrollar un algoritmo que ayude a automatizar la administración de grupos en la FI, enfocándome en la problemática de la asignación y en la búsqueda de la mejora en la distribución de recursos humanos y materiales para lograr un mejor aprovechamiento de la infraestructura de la facultad.

3.3. Justificación y relevancia

La administración de grupos es un problema creciente debido, por un lado, al aumento en la matrícula de estudiantes de las diferentes ingenierías, y por otro, a la falta de recursos económicos para la contratación de profesores, ocasionando un desequilibrio en la oferta-demanda de grupos.

[38] En el capítulo siguiente, se hablará un poco sobre el beneficio y desventaja del uso de la programación concurrente en el algoritmo presentado.

La implementación de un sistema que automatiza la asignación de grupos ayudaría a los administrativos a realizar su trabajo y a su vez dotaría a la FI de un sistema que utiliza técnicas de Inteligencia Artificial que, comparadas con las técnicas clásicas, en este tipo de problema mostrarían una mejora en el resultado y en el proceso que conlleva la administración de grupos, por ejemplo, en la reducción de tiempo invertido en las asignaciones o en el aprovechamiento y distribución de aulas según los horarios planteados.

El desarrollo e implementación de sistemas con enfoques basados en Inteligencia Artificial, como el propuesto, permitirían el fortalecimiento de la infraestructura tecnológica de la FI por lo que es importante impulsar su estudio para resolver problemas reales que se presentan en ella, y a su vez utilizar herramientas emergentes lo suficientemente estables para manejar aplicaciones robustas de este tipo, como el software libre que ha demostrado ser una excelente alternativa al software de pago, por sus múltiples implementaciones en distintos ámbitos y constante desarrollo, corrección y mejora, soportado por una amplia comunidad de programadores alrededor del mundo.

La importancia de proyecto SAG-FI radica en el beneficio general hacia la facultad: los administrativos se verían beneficiados al contar con una herramienta que ayuda a la automatización de la asignación de grupos, la cual es una tarea semestral obligatoria, y la comunidad estudiantil se beneficiaría al contar con múltiples grupos disponibles enfocados a cubrir la demanda.

3.4. Descripción general

El SAG-FI consta de un algoritmo híbrido llamado Simulated Annealing - Squeaky Wheel Optimization (SA-SWO), el cual esta compuesto por un meta-heurístico, SA, que guía a una modificación del algoritmo SWO proponiendo permutaciones de grupos que sirven como entrada a este último. Tales entradas representan la prioridad relativa de cada grupo, siendo

el primer elemento el de mayor prioridad.

Este algoritmo fue seleccionado dado que siempre ofrece *soluciones factibles*, a diferencia de las implementaciones realizadas de manera individual del SA y SWO, donde el primero utilizaba "operaciones" para crear los estados vecinos, entre ellas el operador *cambia-día*, *cambia-hora*, *cambia-salón*, entre otras; y el segundo simulaba a las aulas como "contenedores" de grupos, que a través de mensajes concurrentes, se iban asignando con un enfoque "glotón" que no permitía llegar al óptimo global. Como se mencionó en el capítulo anterior, existen distintos métodos para ofrecer soluciones a los problemas de *timetabling* y poder estudiar todos ellos es prácticamente imposible, por ello se seleccionó uno de los más populares, el SA, y se propuso combinarlo con el SWO para ofrecer un método diferente.

Por otro lado, el objetivo del algoritmo SA-SWO es maximizar el número de asignaciones de las *soluciones válidas* encontradas, es decir, maximizar la cardinalidad de Θ en cada caso. Idealmente, si las restricciones fuertes lo permiten, se desea que $|\Theta| = |G|$. El SA-SWO no garantiza resolver el problema de la **Definición 7**, al verse en la necesidad de relajar el criterio de optimalidad, se transforma en un problema de **semi-optimización** caracterizado por el balance entre la calidad de la solución y el costo de la búsqueda que ésta implica [Pea84].

3.4.1. Algoritmo SA-SWO

El funcionamiento de este algoritmo parte del hecho que el máximo número de grupos a asignar es el número total de grupos, en el caso de que el problema tenga una o varias *soluciones factibles*. Sin embargo, no es posible determinar si el caso de entrada tiene este tipo de solución, por lo que se busca maximizar el número de grupos a asignar.

Inicialmente el algoritmo procesa la entrada y genera tablas con los datos obtenidos, estas estructuras serán dinámicas ya que por cada grupo asignado se deberán de actualizar los datos que estén directamente relacionados y así evitar intersecciones entre disponibilidades

de los diferentes conjuntos. [39]

Hecho lo anterior, inicia la parte del SA dentro del algoritmo del SA-SWO, con ello calcula la solución inicial, la cual será considerada momentáneamente como la mejor. Para construir dicha solución, al igual que las siguientes, se utiliza el SWO. Si la solución es *factible*, termina el algoritmo, en caso contrario calcula la *Tabla de conflictos*, la cual almacena el *conjunto conflicto* de cada grupo, es decir, asocia los grupos que pueden tener horarios comunes en determinadas aulas y después entra a la parte cíclica del SA.

Cuando el SWO termina de construir el estado vecino, la solución generada es comparada con la solución anterior, donde el costo está definido por el número de grupos no asignados. Si el costo de la nueva solución es menor que el de la anterior, el vecino es aceptado, en caso contrario, hay una probabilidad de aceptar esa nueva solución o de conservar la anterior. Independientemente, el algoritmo almacena en memoria aquella solución que presente la menor *energía*, es decir, la solución con menor cantidad de grupos no asignados y ésta es conservada hasta encontrar, si es el caso, una nueva con un costo menor. Esto permite al algoritmo regresar siempre una de las mejores *soluciones válidas* encontradas cuando se llega a la temperatura final.

En caso de que no se haya encontrado alguna *solución factible*, el algoritmo desciende su temperatura mediante la función de *enfriamiento geométrico* definida por $T' = \alpha T$ donde $0 < \alpha < 1$ y continua hasta encontrar una *solución factible* o hasta llegar a la temperatura final. Los siguientes pseudo-códigos muestran la estructura del SA-SWO y la función para calcular la *Tabla de conflictos*:

[39] Los archivos de entrada/salida que son utilizados por SA-SWO serán explicados en la sección *3.4.2. Implementación.*

Algoritmo 2 Algoritmo SA-SWO (SA)

Entrada:

Φ : *La tupla que modela el problema de timetabling*
T_i : *La temperatura inicial*
T_f : *La temperatura final*
N : *El número de iteraciones para el equilibrio térmico*
k : *La constante de normalización*
α : *La constante de enfriamiento*

Salida:

Θ : *Un conjunto de asignaciones*

1: $X \leftarrow$ permutación_aleatoria(G)
2: $(Y, \Theta) \leftarrow SWO(X, \Phi)$
3: $E \leftarrow |G| - |\Theta|$
4: **if** $E = 0$ **then**
5: | **return** Θ
6: **end if**
7: $f_c \leftarrow conflicto(\Phi)$
8: $[\ \Theta_{best}, E_{best}\ ] \leftarrow [\ \Theta, E\ ]$
9: $T \leftarrow T_i$
10: **while** $T_f < T$ **do**
11: | $i \leftarrow 1$
12: | **while** $i \leq N$ **do**
13: | | $X \leftarrow permutación(i, Y, f_c)$
14: | | $(Y, \Theta') \leftarrow SWO(X, \Phi)$
15: | | $E' \leftarrow |G| - |\Theta'|$
16: | | **if** $E' = 0$ **then**
17: | | | **return** Θ'
18: | | **end if**
19: | | **if** $E' < E_{best}$ **then**
20: | | | $[\ \Theta_{best}, E_{best}\ ] \leftarrow [\ \Theta', E'\ ]$
21: | | **end if**
22: | | **if** $(E' < E) \vee$ *con probabilidad* $p = e^{-(\frac{\Delta E}{kT})}$ **then**
23: | | | $[\ \Theta, E\ ] \leftarrow [\ \Theta', E'\ ]$
24: | | **end if**
25: | | $i \leftarrow i + 1$
26: | **end while**
27: | $T \leftarrow \alpha T$
28: **end while**
29: **return** Θ_{best}

Algoritmo 3 Función conflicto

Entrada:

Φ : *La tupla que modela el problema de timetabling*

Salida:

f_c : *Una función que indica el conjunto de grupos conflicto para cada grupo*

```
1:  for all g_1 = (p_1, a_1) ∈ G do
2:      L_1 ← lsort(f_p(p_1))                    ; convierte el conjunto en una lista ordenada
3:      V_1 ← sesiones(L_1, f_s(a_1), f_m(a_1))
4:      f_c(g_1) ← {(g_1, f_p(p_1))}
5:      for all g_2 = (p_2, a_2) ∈ G, g_1 ≠ g_2 do
6:          L_2 ← lsort(f_p(p_2))
7:          V_2 ← sesiones(L_2, f_s(a_2), f_m(a_2))
8:          if set(V_1) ∩ set(V_2) ≠ {} then
9:              f_c(g_1) ← f_c(g_1) ∪ {g_2, (g_2, f_p(p_2))}
10:         end if
11:     end for
12: end for
13: return f_c
```

El SWO originalmente trabaja con tres bloques de manera cíclica: el *Constructor*, el *Analizador* y el *Asignador de Prioridades*. En el algoritmo SA-SWO, inicia por el *Analizador* quien verifica en cada iteración si los grupos que no han sido asignados tienen al menos un aula disponible para ello; después el *Constructor* busca generar un horario válido para un determinado grupo; y finalmente el *Asignador de Prioridades*, reemplazado por otro bloque llamado *Agrupador*, modifica el valor de una bandera que influye en la selección de horarios e indica el término de la construcción del estado vecino. A continuación se muestra el pseudo-código que ilustra el flujo mencionado.

Algoritmo 4 Algoritmo SA-SWO (SWO)

Entrada:

X : *Una permutación formada por los grupos a asignarse*

Φ : *La tupla que modela el problema de timetabling*

Salida:

Y : *Una secuencia cronológica de los grupos asignados*

Θ : *Un conjunto de asignaciones*

```
1: Y ← [ ], X' ← [ ], Θ ← {}
2: DLA ← true
3: do
4:     while X ≠ [ ] do
5:         (G', X') ← analizador(X, X', f_s, f_m)
6:         (X, X', Y, Θ) ← constructor(G', X', Y, Θ)
7:     end while
8:     DLA ← ¬DLA                                   ; bloque "Agrupador"
9:     X ← X'
10:    X' ← [ ]
11: while ¬DLA
12: return Y, Θ
```

El SWO, quien construye los estados vecinos, recibe un ordenamiento de entrada que indica la secuencia con la que se tratarán de asignar los grupos, asigna todos los grupos posibles y al final regresa el ordenamiento solución indicando la secuencia cronológica en la que fueron asignados los grupos, por consiguiente, SWO depende en gran medida de la permutación de entrada.

Los ordenamientos son permutaciones formadas por los grupos a asignarse, dado que todos los grupos son únicos, el total de posibles permutaciones de entrada esta definido por $(n!)$, donde n es el número de grupos. Se proponen tres tipos de ordenamientos de entrada:

- **ordenamiento aleatorio**: es una secuencia de grupos formada de manera aleatoria con distribución uniforme,
- **ordenamiento exploratorio**: genera un número entero positivo p entre 1 y q, siendo q el tamaño del ordenamiento,[40] de manera aleatoria, el cual será la posición del orde-

[40] p garantiza que al menos un grupo será conservado en el próximo ordenamiento.

namiento solución anterior donde será dividido en dos partes: la primera, que incluye al grupo con la posición p, se mantendrá estática y la segunda parte se ordenará de manera aleatoria con distribución uniforme.[41] De esta manera, se conservan las asignaciones de los primeros grupos y se permite explorar otra región del espacio de búsqueda,[42] y

- **ordenamiento glotón**: formado a partir del ordenamiento solución anterior, toma el último grupo que se asignó, obtiene su conjunto conflicto, (previamente calculado y almacenado en la *Tabla de Conflictos*) considera únicamente los grupos del que han sido asignados y de ellos, selecciona el primero que haya sido asignado con disponibilidad original diferente,[43] si no existe alguno, selecciona el primero que se asignó. Después continua de manera similar al *Ordenamiento exploratorio*, pero en este caso p es la posición del primer grupo conflicto seleccionado.

El *Ordenamiento glotón* y el *Ordenamiento exploratorio* dependen de la solución anterior mientras que el *Ordenamiento aleatorio* no requiere de información adicional, por lo que este ordenamiento se utiliza para construir la primera solución, el primer vecino y el estado siguiente cada vez que el SA-SWO desciende la temperatura. Este ordenamiento, generalmente destructivo, permite reiniciar la exploración en una región distinta del espacio de búsqueda. El Algoritmo 5 muestra el pseudo-código correspondiente y la Figura 3.1 ilustra los tipos de ordenamiento explicados anteriormente.

[41] Cuando $p = q$, la segunda parte no incluye grupos y por lo tanto el ordenamiento se mantiene idéntico.
[42] El espacio de búsqueda es el conjunto Θ.
[43] La disponibilidad original es la disponibilidad leída de la entrada.

Algoritmo 5 Función permutación

Entrada:

i : *Número de iteración*

Y : *Una secuencia cronológica de los grupos asignados*

f_c : *Una función que indica el conjunto de grupos conflicto para cada grupo*

Salida:

X : *Una permutación formada por los grupos a asignar*

```
1: if i = 1 then
2: | X ← permutación_aleatoria(Y)
3: else
4: | if probabilidad then          ; menor a 0.55, valor obtenido experimentalmente
5: | | min ← tamaño(Y)                                          ; enfoque "glotón"
6: | | [ g | T ] = Y
7: | | for all y ∈ T do
8: | | | if y ∈ f_c(g) ∧ posición(y) < min ∧ (y, U) ≠ (g, V) then
9: | | | | min ← posición(y)                  ; donde U y V son disponibilidades
10: | | | end if
11: | | end for
12: | else                                                       ; enfoque "exploratorio"
13: | | p ← random(tamaño(Y))                          ; aleatorio entre 1 y tamaño(Y)
14: | end if
15: | [A, B] = split(p, reverse(Y))          ; divide la lista Y a partir de la posición p
16: | X ← [ A | permutación_aleatoria(B) ]
17: end if
18: return X
```

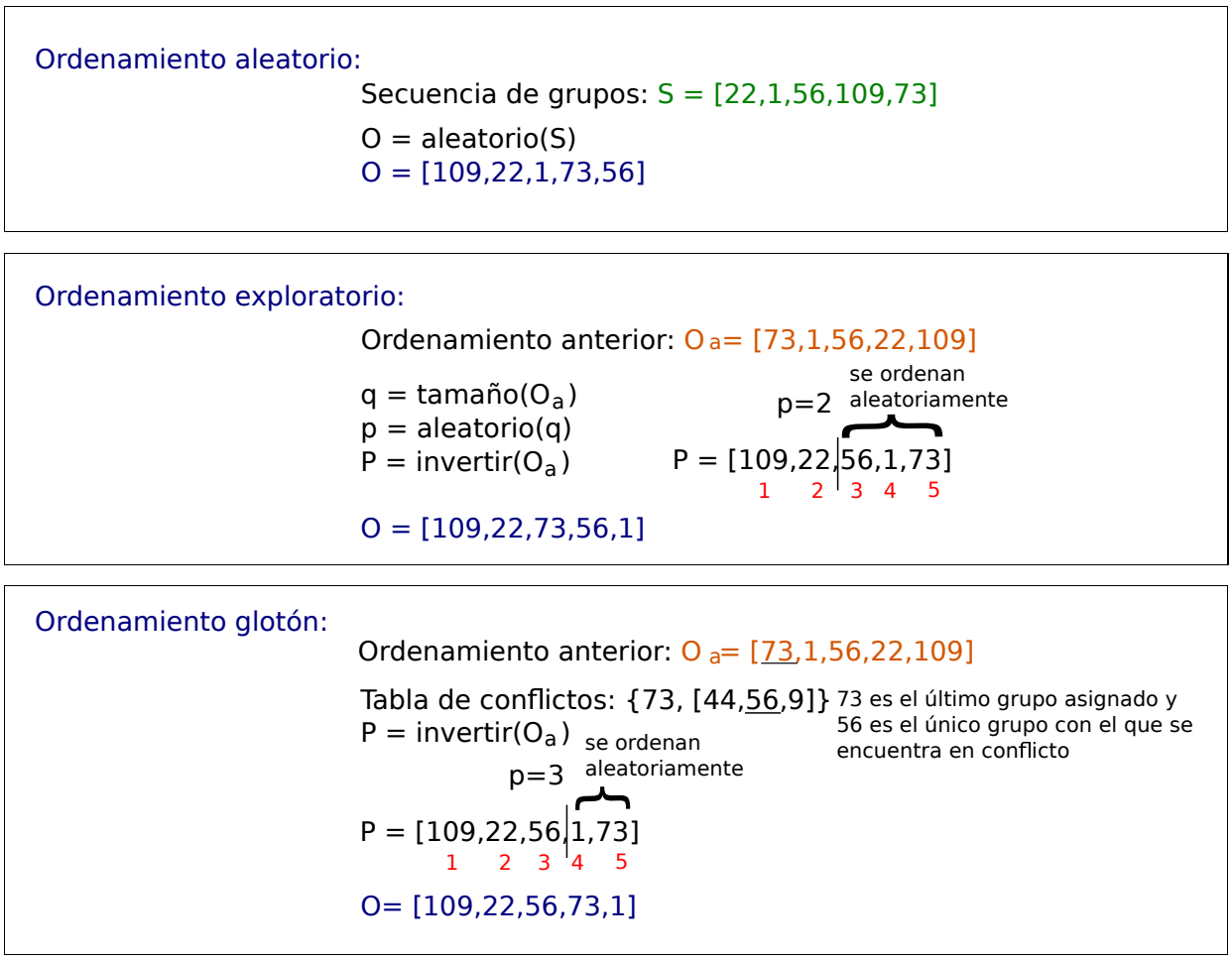

Figura 3.1: Tipos de ordenamientos. El *Ordenamiento aleatorio* toma la secuencia de grupos definida en el problema y ordena sus elementos de manera aleatoria con distribución uniforme; el *Ordenamiento exploratorio* y el *Ordenamiento glotón* dividen su respectivo ordenamiento anterior a partir de una posición p en dos listas: la primera conserva los elementos en el mismo orden y la segunda los ordena de manera aleatoria con distribución uniforme. Para el primer caso, p se obtiene de manera aleatoria y para el segundo depende del *Conjunto conflicto* del último grupo asignado.

A continuación se explica el funcionamiento de cada uno de los bloques:

- **Analizador**: verifica que los grupos del ordenamiento de entrada tengan al menos un aula con la disponibilidad suficiente para ser asignados. Para hacer esa verificación el *Analizador* cuenta los minutos disponibles en las aulas válidas de cada grupo, si el resultado es igual o mayor a los minutos necesarios, el aula se mantiene como disponible, de lo contrario se descarta. Cuando se descartan todas las aulas de un grupo, éste se

elimina del ordenamiento. El Algoritmo 6 muestra el funcionamiento descrito.

Este bloque no asigna ningún *blame factor* a los elementos del problema como el *Analizador* del SWO original, dicho parámetro es reemplazado por los ordenamientos generados por el SA antes de entrar al SWO.

Algoritmo 6 Función analizador

Entrada:

X : *Una permutación formada por los grupos a asignarse*
X' : *Una lista de grupos no asignados*
f_s : *Un conjunto de tuplas que representan funciones de las aulas donde puede impartirse una asignatura*
f_m : *Una función que indica los minutos que dura a la semana una asignatura*

Salida:

G' : *Una lista ordenada de grupos con aulas disponibles para asignarse*
X' : *Una lista de grupos no asignados*

1: $G' \leftarrow [\ ]$
2: $\forall s \in S$ *obtiene el número de minutos* n_s *disponibles del aula* s
3: **for all** $g = (p, a) \in G$ **do**
4: $\quad S' \leftarrow [\ ]$
5: $\quad$ **for all** $s \in f_s(a)$ **do**
6: $\quad\quad$ **if** $f_m(a) \leq n_s$ **then**
7: $\quad\quad\quad S' \leftarrow [\ s \mid S'\ ]$
8: $\quad\quad$ **end if**
9: $\quad$ **end for**
10: $\quad$ **if** $S' = [\ ]$ **then**
11: $\quad\quad X' \leftarrow [\ g \mid X'\ ]$; agrega el elemento a la lista
12: $\quad$ **else**
13: $\quad\quad G' \leftarrow [\ (g, S') \mid G'\ ]$
14: $\quad$ **end if**
15: **end for**
16: **return** G', X'

- **Constructor**: toma un grupo del ordenamiento, después busca un aula disponible dando prioridad a la que en ese momento tenga la mayor cantidad de grupos asignados, esto permitirá primero la distribución de grupos hacia las aulas con mayor ocupación, para dejar a otras con mayor disponibilidad para las *asignaciones especiales* posteriores a la asignación del algoritmo, en caso de ser necesarias.

  El aula seleccionada tiene la disponibilidad suficiente (previamente validada por el *Analizador*) pero el *Constructor* verifica que esa disponibilidad sea igual o superior a la distribución indicada por los esquemas, un parámetro de las asignaturas que indica el número de sesiones y la duración de estas.[44] Cuando la disponibilidad no cubre ningún esquema válido, el grupo es eliminado del ordenamiento, en caso contrario, tomando los horarios en los extremos de la disponibilidad del grupo, se crean todos los horarios posibles y se eliminan, en caso de existir, los duplicados. La Figura 3.2 presenta un par de ejemplos que ilustran el concepto de *esquema*.

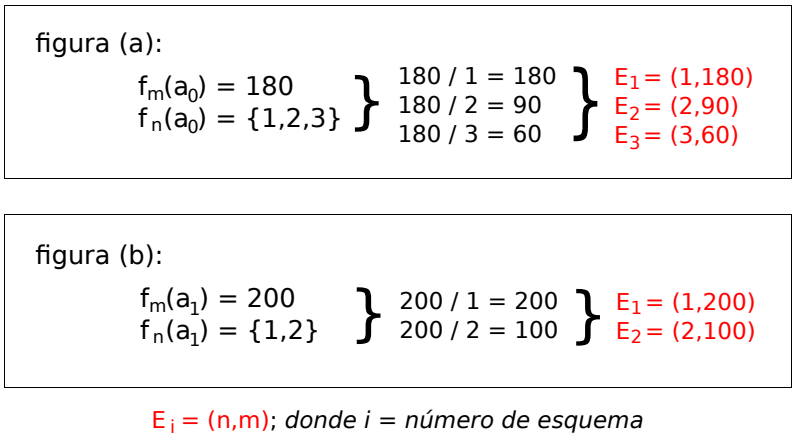

Figura 3.2: Ejemplos de esquemas. Las figuras (a) y (b) muestran la forma en que se cons-truyen los esquemas. Si para la figura (b), $f_n(a_1) = \{1, 2, 3\}$, no sería posible crear un tercer esquema, E_3, dado que $m = 66.\overline{6}$, un número periódico.

[44] Un *esquema* es la tupla (n, m) donde n es el número de sesiones en que se puede impartir una asignatura y m son los minutos de duración para cada una de ellas. m es el cociente de la división entre el número de minutos requeridos a la semana por la asignatura $f_m(a)$ (ver Definición 7) y n, donde $n \in f_n(a), m \in \mathbb{N}^+$. Para cada asignatura, el número de esquemas disponibles es igual al número de elementos dentro del conjunto definido por la función f_n.

Para seleccionar el horario que será asignado, el constructor depende de un procedimiento que funciona de manera similar al proceso de formación de *clusters* de partículas, conocido como *Diffusion-Limited Aggregation (DLA)*,[45] donde una *partícula* se mueve aleatoriamente hasta entrar en contacto con un grupo creciente de partículas. En la versión más sencilla del modelo, una simulación comienza mediante la ocupación de un lugar en el centro de una red (cuadrada o triangular) para representar la *semilla*.[46] Después, es seleccionado un sitio lejano a ésta y una nueva partícula comienza a moverse aleatoriamente desde el sitio seleccionado, si este "caminante aleatorio" se aleja demasiado de la agrupación en crecimiento, finaliza y una nueva partícula con el mismo movimiento inicia. De lo contrario, si el "caminante aleatorio" llega a un sitio que sea el vecino más cercano a otro previamente ocupado, la *partícula* se detiene y el espacio no ocupado se llena, es decir, la *partícula* se agrega. Este proceso se repite muchas veces para simular un proceso de crecimiento [Mea98].

Haciendo la analogía, las *semillas* corresponden a los extremos de los periodos de disponibilidad de las aulas, lo que significa que los primeros grupos en asignarse serán aquellos cuyo horario concuerde con el inicio o el fin de dicho periodo; las *partículas* son las sesiones del grupo por asignarse, que dentro del modelo del algoritmo SA-SWO dependerán de acuerdo al número de sesiones permitido para la asignatura: pueden ser una, dos o tres al mismo tiempo que pero sólo es necesario agregar una de ellas para poder asignar al grupo; y el *caminante aleatorio* es la búsqueda del aula y horario realizada por el *Constructor* para un cierto grupo.

El valor por defecto del DLA es "true", lo que indica que inicialmente los grupos serán asignados en horarios consecutivos, en otras palabras, las asignaciones buscan reducir

[45] El término *Diffusion-Limited* hace referencia a que las partículas se agrupan en bajas concentraciones para que no entren en contacto unas con otras y la estructura crezca de manera uniforme una partícula a la vez [Bou04].

[46] A esta partícula inicial también le conoce como *sitio de crecimiento* o *sitio del núcleo*.

los periodos de *tiempo muerto*.[47]

Un aspecto importante de este bloque es que, a diferencia del *Constructor* del SWO original, no rompe ninguna restricción fuerte. El procedimiento realizado por el *Constructor* es descrito en los siguientes pseudo-códigos.

Algoritmo 7 Función constructor

Entrada:

G' : *Una lista ordenada de grupos con aulas disponibles para asignarse*
X' : *Una lista de grupos no asignados*
Y : *Una secuencia cronológica de los grupos asignados*
Θ : *Un conjunto de asignaciones*

Salida:

X : *Una lista ordenada de grupos a asignar*
X' : *Una lista de grupos no asignados*
Y : *Una secuencia cronológica de los grupos asignados*
Θ : *Un conjunto de asignaciones*

1: $[\ (g, S') \mid T\ ] = G'$; divide la lista G' en la cabeza (g, S') y la cola T
2: $X \leftarrow T$
3: **for all** $s \in S'$ **do** ; obtiene el número de grupos asignados n_s en el aula s
4: $\quad K \leftarrow \{\ \theta \mid \theta_i = (g_i, h_i, s),\ \theta_i \in \Theta\}$
5: $\quad n_s \leftarrow |K|$
6: **end for**
7: $s^* = \underset{s \in A}{\operatorname{argmax}}(n_s)$; selecciona el aula más ocupada
8: $g = (p, a)$
9: $V \leftarrow sesiones(f_p(p), f_n(a), \Phi)$
10: $H \leftarrow horarios(V, s^*)$
11: **if** $H = [\ ]$ **then**
12: $\quad X' \leftarrow [\ g \mid X'\ ]$
13: **else**
14: $\quad [\ h \mid I\ ] = H$
15: $\quad$ *partícula* $\leftarrow (g, h, s^*)$
16: $\quad \theta \leftarrow$ *partícula*
17: $\quad \Theta \leftarrow \Theta \cup \{\theta\}$
18: $\quad Y \leftarrow [\ g \mid Y\ ]$
19: **end if**
20: **return** X, X', Y, Θ

[47] El término *tiempo muerto* hace referencia a lapsos de tiempo disponibles donde no es posible asignar una sesión de ninguna asignatura. Generalmente son intervalos menores a una hora.

Algoritmo 8 Función sesiones

Entrada:

$f_p(p)$: *Un conjunto de tuplas que representan funciones de la disponibilidad de un profesor*

$f_n(a)$: *Un conjunto de tuplas que representan funciones de los números de sesiones en que puede impartirse una asignatura*

Φ : *La tupla que modela el problema de timetabling*

Salida:

V : *Una lista de periodos no necesariamente continuos*

1: $R \leftarrow \text{lsort}(f_p(p))$
2: $W \leftarrow ventanas(R)$
3: $V \leftarrow [\ ]$
4: **for all** $s \in S$ **do**
5: $\quad periodo \leftarrow (\frac{|f_p(p)|}{\max f_n(a)})(\frac{1}{q})$
6: $\quad V \leftarrow discretizar(W, periodo) + V$; concatena las listas
7: **end for**
8: **return** V

Algoritmo 9 Función ventanas

Entrada:

R : *Una lista ordenada que representa una disponibilidad*

Salida:

W : *Un conjunto ordenado de listas de dos elementos, los cuales indican el inicio y el fin de un lapso de tiempo*

1: $W \leftarrow [\ ]$
2: $[\ H_1 \mid T_1\ ] = R$
3: **if** $T_1 = [\ ]$ **then**
4: $\quad W \leftarrow R$
5: **else**
6: $\quad i \leftarrow 1$
7: $\quad$ **while** $T_i \neq [\ ]$ **do**
8: $\quad\quad [\ H_{i+1} \mid T_{i+1}\ ] = T_i$
9: $\quad\quad$ **if** $H_i + 1 \neq H_{i+1}$ **then**
10: $\quad\quad\quad W \leftarrow [\ [H_1, H_i] \mid ventanas(T_i)\ ]$
11: $\quad\quad$ **else**
12: $\quad\quad\quad i \leftarrow i + 1$
13: $\quad\quad$ **end if**
14: $\quad$ **end while**
15: **end if**
16: **return** W

Algoritmo 10 Función discretizar

Entrada:

W : *Un conjunto ordenado de listas de dos elementos los cuales indican el inicio y el fin de un lapso de tiempo*

periodo : *Un intervalo de tiempo que representa una sesión*

Salida:

V : *Una lista de periodos no necesariamente continuos*

```
1: [ H | T ] = W
2: [ H_1 | H_2 ] = H
3: if H_2 − H_1 > periodo then
4:  |  V ← [ [ H_1 + periodo ] | discretizar([ H_1 + periodo + 1, H_2 ] | T ], periodo)
5: else if H_2 − H_1 = periodo then
6:  |  V ← [ [ H_1 + periodo ] | discretizar(T, periodo) ]
7: else
8:  |  V ← discretizar(T, periodo)
9: end if
10: return V
```

Algoritmo 11 Función horarios

Entrada:

V : *Una lista de posibles horarios para un grupo*

s^* : *La mejor aula candidata para la asignación*

Salida:

H : *Una lista de posibles horarios para un grupo seleccionado*

```
1: H ← [ ]
2: Obtiene el horario reservado R_{s*} del aula s*
3: for all v ∈ V do
4:  |  aceptado ← agregar(v, R_{s*})
5:  |  if aceptado then
6:  |  |  H ← [ v | H ]
7:  |  end if
8: end for
9: return H
```

Algoritmo 12 Función agregar

Entrada:

v : *Un horario candidato para el grupo*

R_{s^*} : *El horario reservado del aula*

Salida:

aceptado : *Variable booleana que indica si el horario del grupo es aceptado*

```
1: if R_{s*} = [ ] then
2:     aceptado ← false
3: else
4:     [ h | T ] = v
5:     [ b | C ] = R_{s*}
6:     [ h_i, h_f ] = h
7:     [ b_i, b_f ] = b
8:     if (h_i = b_f + 1) ∨ (h_f = b_i − 1) then
9:         aceptado ← true
10:    else if agregar([h], C) then
11:        aceptado ← true
12:    else
13:        aceptado ← agregar(T, R_{s*})
14:    end if
15: end if
16: return aceptado
```

- **Agrupador**: cuando todos los grupos del ordenamiento se han intentado asignar por primera vez, la bandera del DLA cambia su valor a "false", lo que significa que los grupos que no han sido asignados formarán un nuevo ordenamiento y se tratarán de asignar sin importar si pueden agregarse a alguna agrupación de sesiones o no. Cuando se entra al *Agrupador* por segunda ocasión, significa que quedaron grupos que en definitiva no pueden ser asignados en la solución construida y termina el SWO.

La Figura 3.3 muestra el flujo de los bloques del SWO del algoritmo SA-SWO.

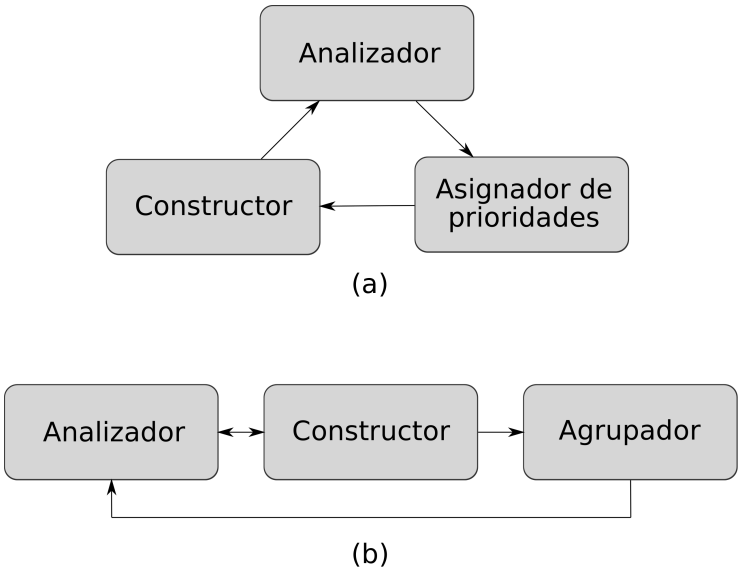

Figura 3.3: Bloques del algoritmo SWO. La figura (a) muestra los bloques que forma el algoritmo SWO. La figura (b) muestra la modificación de los bloques del SWO dentro del algoritmo SA-SWO utilizados para formar las asignaciones (estados del SA).

3.4.2. Implementación

El algoritmo SA-SWO es un conjunto de módulos[48] programados en Erlang y como entrada recibe un archivo en formato JSON constituido por un objeto con los siguiente pares *nombre/valor*:

- **horario_disponible**: comprende el horario donde es posible asignar grupos. Su valor se forma por un arreglo de un objeto que incluye dos pares: el primero con el nombre *"dias"* y cuyo valor es un arreglo de enteros en secuencia ascendente del cero al cuatro, donde el cero representa el día lunes, el uno al martes y así sucesivamente hasta el cuatro que representa el viernes; el segundo par tiene por nombre *"lapsos"* y como valor tiene un arreglo formado por un objeto a la vez formado por dos pares con nombre *"inicio"* y *"fin"* donde las horas de inicio y término del horario son sus valores representados por cadenas, por ejemplo: *"inicio": "07:00" y "fin": "22:15"*,

- **profesores**: su valor es un arreglo de objetos definidos por tres pares: el primero tiene como nombre *"id"* y como valor un entero que representa su identificador; el segundo tiene por nombre *"id2"* y como valor una cadena de caracteres que indica el nombre del profesor; el último par tiene por nombre *"disp"* y su valor es un arreglo de objetos similar al valor de **horario_disponible**,

- **grupos**: su valor consta de un arreglo de objetos definidos por cuatro pares cuyos nombres son *"id"*, *"id2"*, *"profesor"* y *"asignatura"*. A diferencia de los demás valores, que son números enteros que representan el identificador del grupo, profesor y asignatura respectivamente, el valor de *"id2"* es una cadena de caracteres con el nombre común del grupo, por ejemplo *"1306-4"* representa al grupo número cuatro de la asignatura 1306, Ecuaciones Diferenciales,

[48] Un *módulo* es la unidad básica de código en Erlang y almacena todas las funciones en archivos con extensión *.erl* [Arm07].

- **aulas**: su valor es un arreglo de objetos formados por dos pares: el primero tiene por nombre *"id"*, su valor es un entero que simboliza su identificador y el segundo tiene por nombre *"id2"*, su valor es una cadena de caracteres que representa el nombre del aula, por ejemplo, *"A201"* simboliza el espacio "01" del nivel "2" del edificio "A", [49] y

- **asignaturas**: su valor es un arreglo de objetos formados por cinco pares: los primeros dos pares representan los identificadores, *"id"* y *"id2"*, el primero con un entero y el segundo con una cadena de caracteres; el tercer par tiene como nombre *"minutos"* y como valor un entero que representa el número de minutos a la semana que requiere la asignatura; y el cuarto y quinto par tienen como nombre *"aulas"* y *"sesiones"* donde ambos valores son representados por un arreglo de números enteros, para el cuarto par representan el identificador y para el quinto, el número de sesiones en que se puede impartir la asignatura.

Como salida, el algoritmo SA-SWO genera un archivo JSON con la solución encontrada en la prueba. Los pares *nombre/valor* que forman el objeto de dicho archivo son:

- **grupos_no_asignados**: Su valor es un objeto formado por pares con el nombre *"id"* y con valores de números enteros representando el identificador de los grupos, en caso de existir, que no fueron asignados,

- **profesores**: Su valor contiene la misma estructura que *"profesores"* en el archivo de entrada, y

- **aulas**: Su valor es semejante a la de *"aulas"* en el archivo de entrada, a diferencia de que posee un par adicional con nombre *"horario"* y valor un arreglo con estructura similar a *"horario_disponible"* también del archivo de entrada, que indica los horarios de los distintos grupos que fueron asignados en una determinada aula.

[49] `http://servacad.ingenieria.unam.mx/salones/`

Todos los nombres de los pares utilizados en ambos archivos son cadenas de caracteres, esto con el fin de facilitar la interpretación de sus respectivos valores. El Cuadro 3.1 muestra el esquema del archivo de entrada y el Cuadro 3.2 muestra el de salida.

Cabe mencionar que los archivos en formato JSON permitirían la integración con otros sistemas ya que utilizan un formato ligero de intercambio de datos y actualmente muchos lenguajes de programación incluyen librerías para interpretarlos.[50]

input.json

```
{
  "horario_disponible": [{ "dias":arreglo_enteros,
    "lapsos":[{ "inicio":cadena,"fin":cadena }]}],
  "profesores": [
    { "id":entero, "id2":cadena,
      "disp":[{ "dias":arreglo_enteros,
      "lapsos":[{ "inicio":cadena,"fin":cadena }], ... }] },
    ...
  ],
  "grupos": [
    { "id":entero, "id2":cadena, "profesor":entero,
      "asignatura":entero },
    ...
  ],
  "aulas": [
    { "id":entero, "id2":cadena },
    ...
  ],
  "asignaturas": [
    { "id":entero, "id2":cadena, "minutos":entero,
      "aulas":arreglo_enteros, "sesiones":arreglo_enteros },
    ...
  ]
}
```

Cuadro 3.1: Parámetros del archivo JSON de entrada.

[50]Introducing JSON, `http://www.json.org`

output.json

```
{
  "grupos_no_asignados": {
    "id":entero, ...
  },
  "profesores": [
    { "id":entero, "id2":cadena,
      "disp":[{ "dias":arreglo_enteros,
      "lapsos":[{ "inicio":cadena,"fin":cadena }], ... }] },
    ...
  ],
  "aulas": [
    { "id":entero, "id2":cadena, "horario_gpos":[{ "id":entero,
      "id2":cadena, "horario":[{ "dias":arreglo_enteros,
      "lapsos":[{ "inicio":cadena,"fin":cadena }] }, ... ] }] },
    ...
  ]
}
```

Cuadro 3.2: Parámetros del archivo JSON de salida.

3.5. Alcance y limitaciones

El alcance de este proyecto consiste en el diseño del algoritmo para mejorar la distribución de grupos como un módulo independiente que permita su fácil integración con las herramientas actuales.

Los horarios de los grupos que algoritmo SA-SWO puede asignar, siempre y cuando las restricciones fuertes lo permitan, cumplen con el siguiente modelo:

- los horarios son formados en base a los *esquemas* de cada grupo,
- todas las sesiones de un grupo se asignan en la misma aula,
- si el grupo tiene más de una sesión, éstas se asignan a la misma hora pero en diferente día,[51]
- los horarios únicamente se asignan dentro de la disponibilidad del grupo, formada por

[51] Las múltiples sesiones de un mismo grupo se crean con un día libre entre ellas, es decir, para dos sesiones los días que pueden asignarse son lunes-miércoles, martes-jueves o miércoles-viernes; para tres sesiones solamente se pueden asignar lunes-miércoles-viernes.

la intersección de la disponibilidad del profesor y la del aula en el momento que se intente asignar el grupo[52] (la Figura 3.4 muestra algunos casos que puede tomar la disponibilidad de un grupo), y

- las sesiones únicamente son asignadas en las aulas especificadas como válidas para la asignatura en cuestión.[53]

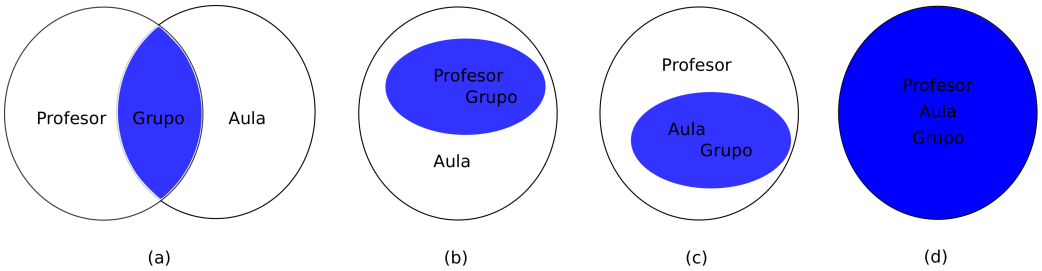

Figura 3.4: Casos de disponibilidades para un grupo. En la figura (a) la disponibilidad del grupo es la intersección entre la disponibilidad de profesor y la del aula; la figura (b) muestra la disponibilidad del profesor completamente contenida dentro de la disponibilidad del aula, por lo tanto la disponibilidad del profesor y la del grupo es la misma; en la figura (c) la disponibilidad del grupo y la del aula son equivalentes debido a que la disponibilidad del aula es un subconjunto de la disponibilidad del profesor; y en la figura (d) todas las disponibilidades son iguales. Más adelante, se especificará que el caso de la figura (b) es utilizado en los *Casos A* y *B* y el caso de la figura (d) es usado en el *Caso C*.

Los conflictos referentes a horarios que el algoritmo SA-SWO puede resolver son:

- *intersección entre grupos de un profesor*: es el caso cuando dos grupos de un profesor comparten un horario en común. Esto puede suceder cuando el profesor pertenece a más de una división académica. Para evitar este problema serían conveniente realizar las asignaciones tomando en cuenta todas las divisiones a la vez y no individualmente como se realiza actualmente, debido a que se necesita manejar información que afecta a más de una división,

- *intersección entre sesiones de dos diferentes grupos dentro de un aula*: sucede cuando el inicio de una sesión no coincide con el termino de otra y se anticipa a ésta,

[52] La intersección de disponibilidades son valores dinámicos que se actualizan en caso de que la asignación anterior tome algún subconjunto de las disponibilidades de otros profesores y/o aulas. Con ello se evitan las intersecciones entre grupos.

[53] Indicadas como parámetro en el archivo JSON de entrada.

- *correcta asignación de horas a cada grupo*: basándose en los minutos de duración, se dividen en sesiones de misma duración y se forman los diferentes esquemas disponibles, y

- *grupos sin aula definida*: ocurre en caso de que ningún aula tenga la disponibilidad requerida para asignar al grupo.[54]

Las limitaciones del algoritmo SA-SWO son asignaciones que requieren del criterio del personal administrativo y en la práctica pueden llegar a romper restricciones fuertes pero resultan benéficas a ciertos grupos y/o profesores. A este conjunto se les llama *asignaciones especiales* y son formadas por casos particulares que deben realizarse después de haber realizado la asignación automática intercambiando los grupos en los horarios deseados. Agregar los casos particulares antes de la asignación automática podría dificultar la búsqueda de soluciones ya que el espacio se encontraría en algunas regiones restringido lo cual afecta directamente a la disponibilidad de los demás grupos en las aulas ocupadas.

Las *asignaciones especiales* sirven para resolver conflictos a nivel administrativo. Algunos casos especiales se enlistan a continuación:

- *grupos equivalentes*: son grupos formados por dos o más asignaturas que por la semejanza en contenido entre los planes de estudio (70 % o más) pueden considerarse equivalentes,

- *grupos simultáneos*: son los casos cuando un profesor imparte clase a dos o más grupos al mismo tiempo en una misma aula,

- *grupos con más de un profesor*: suceden cuando en un grupo de múltiples sesiones un profesor se encarga de impartir una sesión y otro profesor las restantes,

[54]Es importante recalcar que no es posible garantizar la existencia de la relación unívoca entre un grupo y una aula que simbolice la asignación del grupo, debido a las múltiples restricciones involucradas.

- *grupos con más de un aula*: son aquellos grupos donde una sesión se imparte en un aula y las restantes utilizan una distinta,
- *grupos con sesiones de diferente duración*: son grupos donde las sesiones no son equivalentes entre si en el tiempo de duración,
- *grupos con diferentes horarios entre sesiones*: en estos casos, los horarios de las sesiones tienen diferentes tiempos de inicio o de finalización,
- *grupos con horas distintas a las especificadas en el temario*: son los casos donde la suma del tiempo de las sesiones sobrepasa las horas establecidas en el temario,
- *grupos con más de una aula a la vez*: suceden cuando un grupo tiene reservado dos aulas a la vez aunque en realidad solamente utilice una,
- *grupos con más de un profesor a la vez*: ocurre cuando dos profesores imparten la misma sesión, y
- alguna combinación de los casos anteriores[55].

A pesar de que los *casos especiales* son un pequeño porcentaje del total de grupos a asignar, vale la pena manejarlos de manera independiente para no volver más restrictivo el problema.

3.6. Resultados esperados

El resultado prioritario a esperar es el de ofrecer una herramienta efectiva y práctica que permita ofrecer soluciones al problema de asignación de grupos que enfrenta la FI, mostrando una mejora contra el método de distribución actual y con ello optimizar el proceso

[55]La duplicidad de cierto elemento como el profesor o el aula puede incrementarse y considerarse de igual manera como conflicto.

administrativo que conlleva. Por otro lado, se espera que el proyecto sirva como antecedente y fomento para el futuro estudio y desarrollo de sistemas para la facultad usando técnicas de Inteligencia Artificial que superen a las técnicas clásicas y doten a la misma de una infraestructura más sólida[56].

3.7. Casos de estudio

Para poder evaluar el funcionamiento del algoritmo SA-SWO, se plantearon tres casos diferentes nombrados *Caso A*, *Caso B* y *Caso C*, los cuales sirven como entrada para el algoritmo y permiten estudiarlo con distintas variantes.

3.7.1. Caso A: 50 Grupos

El *Caso A* fue diseñado a partir de una solución factible,[57] donde la principal característica consiste en tener todos los horarios de los grupos asignados de manera consecutiva, es decir, no hay espacios de tiempo disponibles entre sesiones además de que la ocupación de las aulas es al 100 %. Para el caso, las variables se restringieron de la siguiente manera:

- dos aulas con horario disponible de 07:00 a 22:00 horas de lunes a viernes,[58]
- diez profesores con disponibilidad de lunes a viernes, de los cuales, cuatro de ellos de 07:00 a 15:00 horas; dos de 10:30 a 18:30 horas y cuatro de 14:00 a 22:00 horas,

[56]Un ejemplo de un sistema favorable para la Facultad consistiría en diseñar una base de datos inteligente para los catálogos de las bibliotecas, los cuales muchas veces regresan resultados pobres cuando no encuentran una incidencia exacta de las consultas realizadas. La utilización de algoritmos de Inteligencia Artificial, Minería de Datos y Aprendizaje Máquina ofrecerían mejores resultados y lo reflejarían en un servicio constantemente requerido.

[57]Ver *Apéndice A. Solución factible base para el Caso A.*

[58]Al considerar al horario de las aulas como el máximo posible, las disponibilidades tanto de los profesores como de los grupos estarán contenidas dentro de este horario, por lo tanto, la disponibilidad de un grupo será igual a la de su respectivo profesor. Gráficamente, se puede apreciar en el caso (b) de la Figura 3.4.

- cinco asignaturas de 180 minutos de duración que pueden impartirse en cualquiera de las dos aulas en una o dos sesiones (una sesión de tres horas o dos sesiones de hora y media), y
- 50 grupos: cinco pertenecientes a cada profesor y cada uno de ellos de diferente asignatura.[59]

El tamaño del espacio de búsqueda del *Caso A* puede expresarse como:

$$\textit{número de grupos} \left(\frac{\frac{\textit{horario disponible de las aulas}}{\textit{resolución}}}{\frac{\textit{duración de las asignaturas}}{\textit{resolución}}} \right) \textit{número de salones}$$

$$50\binom{915}{36}2 \approx 5.4658 \times 10^{66} \text{ posibles soluciones candidatas}$$

Los parámetros de la parte del SA son los siguientes:[60]

- Temperatura inicial (T_i): 3
- Temperatura final (T_f): 1
- Constante de normalización (k): 1
- Número de configuraciones (N): 5
- Constante de enfriamiento (α): 0.95

En particular, este caso se asemeja a la asignación de exámenes departamentales que tenía lugar los días sábados en la División de Ciencias Básicas, donde todos los exámenes se aplicaban en sesiones de la misma duración y dichas sesiones tenían horario consecutivo. La asignación de los laboratorios de las asignaturas de "Estática" y "Cinemática y Dinámica"

[59] Los datos mencionados se encuentran detallados en el archivo JSON de entrada *50.json*.

[60] Para determinar los parámetros, se ejecutaron con anterioridad algunas pruebas donde se analizó la probabilidad generada entre iteraciones y el tiempo que tarda en encontrar alguna solución factible o en llegar a la temperatura final.

era otro caso donde el número de aulas era restringido (propiamente eran dos laboratorios), todas las sesiones tenían la misma duración, eran únicas para cada grupo y ocurrían cada 15 días; además de que la mayoría de los horarios eran consecutivos para dar cavidad a la mayor cantidad de grupos posibles.

3.7.2. Caso B: División de Ciencias Básicas

El *Caso B* es un subconjunto de la asignación presentada en el semestre 2011-1, donde se tomaron los grupos asignados de la División de Ciencias Básicas, básicamente por ser una de las divisiones académicas con mayor matrícula y con menor número de casos que requieren de *asignaciones especiales*. Sin embargo, de los datos obtenidos se depuraron los que presentaron información incompleta (sin aula definida y sin profesor asignado) e incorrecta (grupos con intersección entre sus horarios, ya sean del profesor o del aula), quedando de la siguiente manera:

- 52 aulas con horario disponible de 07:00 a 22:15 de lunes a viernes,
- 289 profesores con distintas disponibilidades,[61]
- 30 asignaturas de 270, 240 y 120 minutos de duración con diferentes aulas donde pueden impartirse[62], y
- 575 grupos.[63]

De esta manera el *Caso B*, al igual que el *Caso A*, parte de una *solución factible*.

El tamaño del espacio de búsqueda del *Caso B* puede expresarse como:

$$413\binom{915}{54}46 + 35\binom{915}{48}25 + 127\binom{915}{24}5 \approx 1.3781 \times 10^{92} \text{ posibles soluciones candidatas}$$

[61] Las disponibilidades de los profesores son el resultado de la unión de sus horarios especificados en la asignación.

[62] Las aulas se limitaron a las ocupadas en la asignación presentada por los administrativos.

[63] La información detallada se encuentra en el archivo JSON de entrada *DCB.json*.

Los parámetros de la parte del SA son los siguientes:

- Temperatura inicial (T_i): 15
- Temperatura final (T_f): 7.5
- Constante de normalización (k): 1
- Número de configuraciones (N): 3
- Constante de enfriamiento (α): 0.90

3.7.3. Caso C: hdtt4-8

El *Caso C* consiste en 5 problemas propuestos por la Profesora Kate Smith-Miles[64] los cuales han sido utilizados como modelos de prueba en diversos estudios [Smi99] [Smi03] [Car04] [Ran01] [Liu09] [Pim13] y forman parte del repositorio de *OR-Library*. [65] Estos problemas: hdtt4, hdtt5, hdtt6, hdtt7 y hdtt8, cuyos nombres hacen referencia al término *hard timetabling*, fueron diseñados para ser totalmente restrictivos, es decir, las *soluciones factibles* ocupan toda la disponibilidad de los profesores, en cada *ventana de tiempo* debe ser asignado un único grupo (no hay intersecciones de ningún tipo) y las aulas son ocupadas al 100 % (no queda ninguna ventana disponible, como en las *soluciones factibles* del *Caso A*).

Gracias a las características anteriores y a la ausencia de *restricciones débiles*, se pudieron modelar los problemas mencionados como entradas para el algoritmo SA-SWO. El cuadro 3.3 muestra las elementos de cada una.

Con el valor de las *ventanas de tiempo* se formaron las disponibilidades iniciales de lunes a viernes de 07:00 a 13:00 horas para los profesores, las aulas y los grupos, tal como el caso (d) de la Figura 3.4. Adicionalmente, el valor de la *resolución* (q) se modificó a 60, así, por cada día hay 6 *ventanas de tiempo* con duración de una hora cada una.

[64] `http://users.monash.edu/~ksmiles/`
[65] `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/tableinfo.html`

Problema	Profesores	Aulas	Ventanas de tiempo	Asignaturas	Grupos
hdtt4	4	4	30	4	120
hdtt5	5	5	30	5	150
hdtt6	6	6	30	6	180
hdtt7	7	7	30	7	210
hdtt8	8	8	30	8	240

Cuadro 3.3: Caso C. Elementos de los problemas.

El Cuadro 3.4 muestra los respectivos tamaños de los espacios de búsqueda de cada caso.

Problema	Posibles soluciones candidatas
hdtt4	14 000
hdtt5	22 500
hdtt6	32 400
hdtt7	44 100
hdtt8	57 600

Cuadro 3.4: Caso C. Tamaños de los espacios de búsqueda.

El Cuadro 3.5 muestra los parámetros de la parte del SA empleados en cada caso.

Problema	$\mathbf{T_i}$	$\mathbf{T_f}$	k	N	$\boldsymbol{\alpha}$
hdtt4	9	1	1	5	0.94
hdtt5	9	1	1	6	0.96
hdtt6	9	1	1	7	0.98
hdtt7	12	1	1	5	0.98
hdtt8	13.5	1	1	5	0.98

Cuadro 3.5: Caso C. Parámetros utilizados.

donde:

- $\mathbf{T_i}$: Temperatura inicial,
- $\mathbf{T_f}$: Temperatura final,
- **k**: Constante de normalización,
- **N**: Número de configuraciones, y
- $\boldsymbol{\alpha}$: Constante de enfriamiento

Capítulo 4

Resultados y observaciones

Las pruebas para evaluar al algoritmo SA-SWO fueron realizadas en dos equipos de cómputo diferentes, los cuales cuentan principalmente con las siguientes especificaciones:[66]

- *Equipo 1*: Procesador AMD Quad-Core A8-3520M a 1.6 GHz / 2.5 GHz[67] y Memoria SDRAM DDR3 de 8 GB.
- *Equipo 2*: Procesador AMD AthlonII X2 220 a 2.8 GHz y Memoria SDRAM DDR3 de 3 GB.

Para el *Caso A* se ejecutaron 100 pruebas (50 por equipo) y para los *Casos B y C* fueron un total de 40 pruebas (20 por equipo). Las siguientes secciones del capítulo muestran algunas gráficas y cuadros comparativos con los resultados obtenidos para cada caso.

[66]Para estas pruebas, la capacidad del procesador y de la memoria RAM son parámetros suficientes para medir el desempeño del algoritmo en tiempo de ejecución y recursos requeridos.

[67]La frecuencia base es a 1.6 GHz y la máxima de 2.5 GHz es alcanzada solamente bajo alta demanda y en pequeños lapsos de tiempo.

4.1. Caso A: 50 Grupos

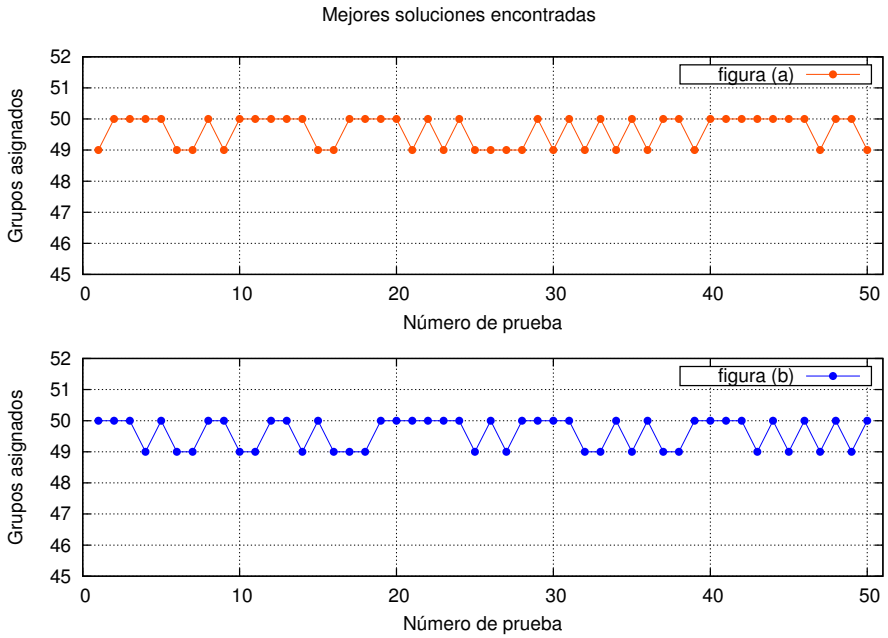

Figura 4.1: Mejores soluciones encontradas. La figura (a) muestra las soluciones de las pruebas en el Equipo 1 y la figura (b) las del Equipo 2.

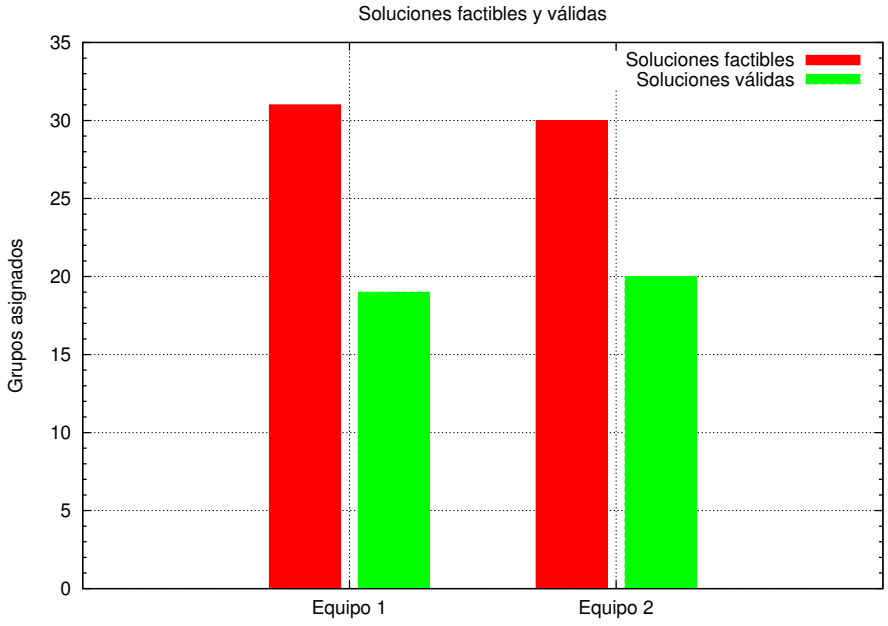

Figura 4.2: Soluciones factibles y válidas. La gráfica muestra el número obtenido de los distintos tipos de soluciones encontradas en las pruebas de ambos equipos.

Las gráficas anteriores muestran que el 61 % de las soluciones obtenidas fueron *soluciones factibles* mientras que el restante 39 % fue de *soluciones válidas* y todas ellas con 49 grupos asignados (98 % del total de grupos), esto quiere decir que en 39 pruebas hubo un grupo en cada una de ellas que no pudo ser asignado. La Figura 4.3 muestra tales grupos y su frecuencia.

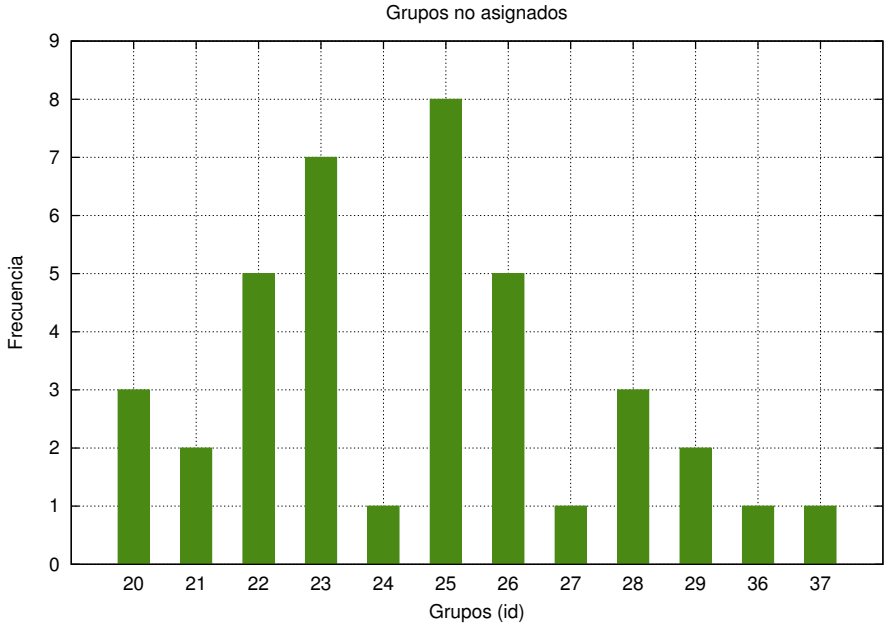

Figura 4.3: Grupos no asignados. La figura muestra la frecuencia de los grupos que no fueron asignados en las pruebas con soluciones válidas en ambos equipos.

Los grupos que se muestran en la gráfica se pueden dividir en dos conjuntos de acuerdo a sus disponibilidades: el primero estaría formado por los grupos con *id* del 20 al 29 con disponibilidad de 10:30 a 18:30 horas y el segundo por los grupos con *id* 36 y 37 con disponibilidad de 14:00 a 22:00 horas. El primer conjunto cuenta con todos los grupos del caso que cumplen dicha disponibilidad, este comportamiento es debido a la intersección de su horario de disponibilidad con sus semejantes de 07:00 a 15:00 horas y de 14:00 a 22:00 horas, por lo tanto, todos los grupos de este conjunto están en conflicto con el resto de los grupos del caso, razón por la que siempre serán los últimos en tratar de asignarse. Para el segundo conjunto, la poca frecuencia de los grupos es debido al comportamiento anterior, inclusive cualquier

otro grupo con disponibilidad distinta a la de 10:30 a 18:30 horas, como los grupos con *id* 30 o 34 podrían de igual manera no asignarse y su frecuencia sería también baja.

Al analizar los archivos de salida, se encontró que algunas aulas con horario disponible tenían una distribución de ese horario dividida en periodos de "tiempo muerto" y en otras tal distribución no permite crear dos sesiones en el mismo horario y en diferente día. En caso de ser necesario, la asignación manual podría resolver el conflicto dándole a ese único grupo un horario con sesiones a diferente día y con horario distinto.

Un parámetro importante a considerar para evaluar el desempeño del algoritmo es el tiempo que tarda en ofrecer una solución. La Figura 4.4 muestra los tiempos de ejecución de cada prueba, la Figura 4.5 presenta exclusivamente los tiempos obtenidos por las soluciones factibles y la Figura 4.6 los tiempos que tarda el algoritmo en alcanzar la temperatura de paro.

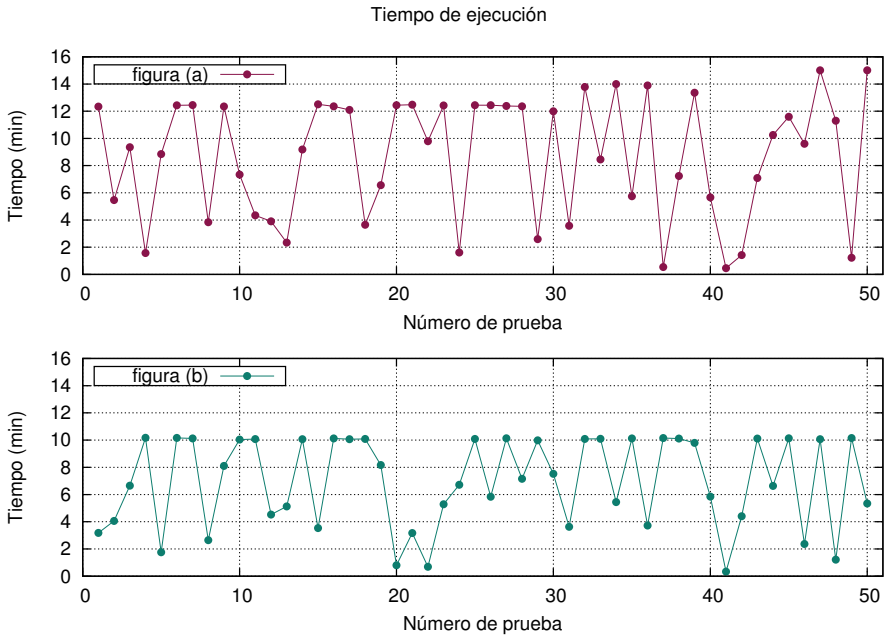

Figura 4.4: Tiempo de ejecución. La figura (a) muestra los tiempos de ejecución por prueba en el Equipo 1; la figura (b) lo muestra para el Equipo 2.

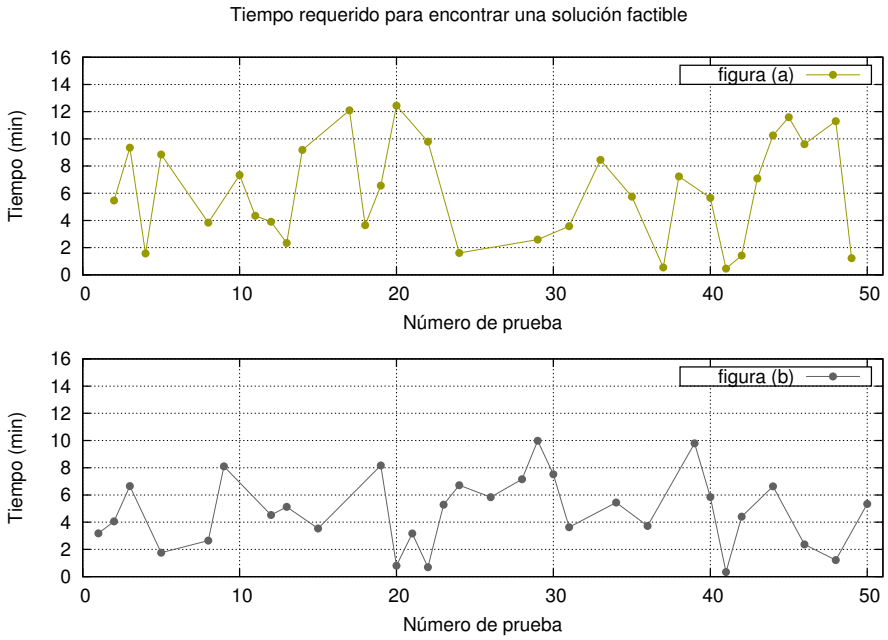

Figura 4.5: Tiempo requerido para encontrar una solución factible. Las gráficas muestran las pruebas que lograron obtener una solución factible. La figura (a) muestra las del Equipo 1 mientras que la figura (b) muestra las del Equipo 2.

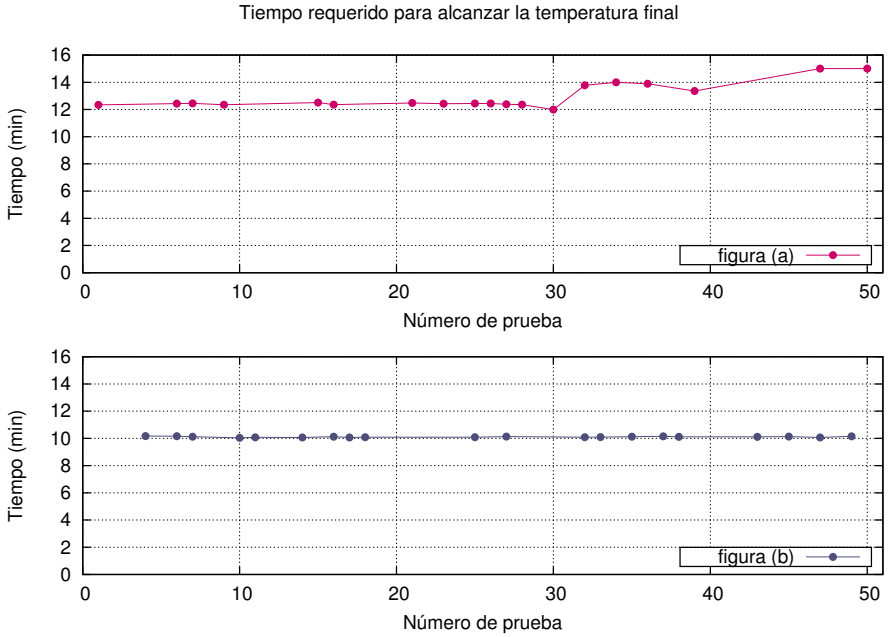

Figura 4.6: Tiempo requerido para alcanzar la temperatura final. Las gráficas muestran las pruebas que lograron una solución válida. Nuevamente la figura (a) muestra los resultados del Equipo 1 y la figura (b) los del Equipo 2.

Los tiempos logrados cuando se llega a una *solución factible* son variados, esto se debe a las probabilidades generadas que permiten crear diferentes permutaciones que darán lugar a los ordenamientos en que se tratarán de asignar los grupos. De acuerdo a las gráficas, para el *Equipo 1* el tiempo promedio fue de **6.10 minutos** y para el *Equipo 2* fue de **4.79 minutos**. La diferencia entre ambos promedios fue de **1.31 minutos**.

Por otro lado, los tiempos de la Figura 4.6 son más uniformes, el promedio para llegar a la temperatura final fue de **12.94 minutos** para el *Equipo 1* y de **10.11 minutos** para el *Equipo 2*. La diferencia entre ambos promedios fue de **2.83 minutos**.

Las siguientes figuras comparan el consumo de recursos en los equipos de cómputo utilizados. Los valores obtenidos son la diferencia entre la carga "normal" del equipo (la memoria y procesador utilizados por procesos inicializados por defecto) y la carga generada mientras se ejecutaban pruebas secuenciales del algoritmo separadas por intervalos de diez minutos durante una hora.

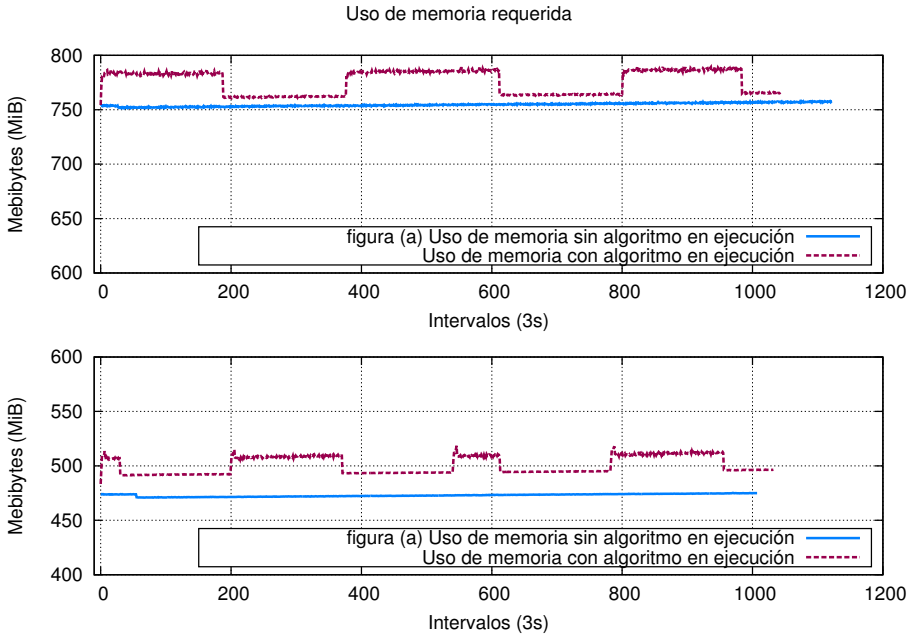

Figura 4.7: Uso de memoria requerida. La figura (a) muestra el consumo de memoria en el Equipo 1 y la figura (b) muestra el consumo en el Equipo 2.

Equipo	Consumo “normal” (MiB)	Consumo con algoritmo en ejecución(MiB)	Memoria requerida (MiB)
1	754.69	783.49	28.80
2	472.98	508.58	35.60

Cuadro 4.1: Caso A. Promedios de consumo de memoria. Datos duros obtenidos de la información presentada en las gráficas de la Figura 4.7.

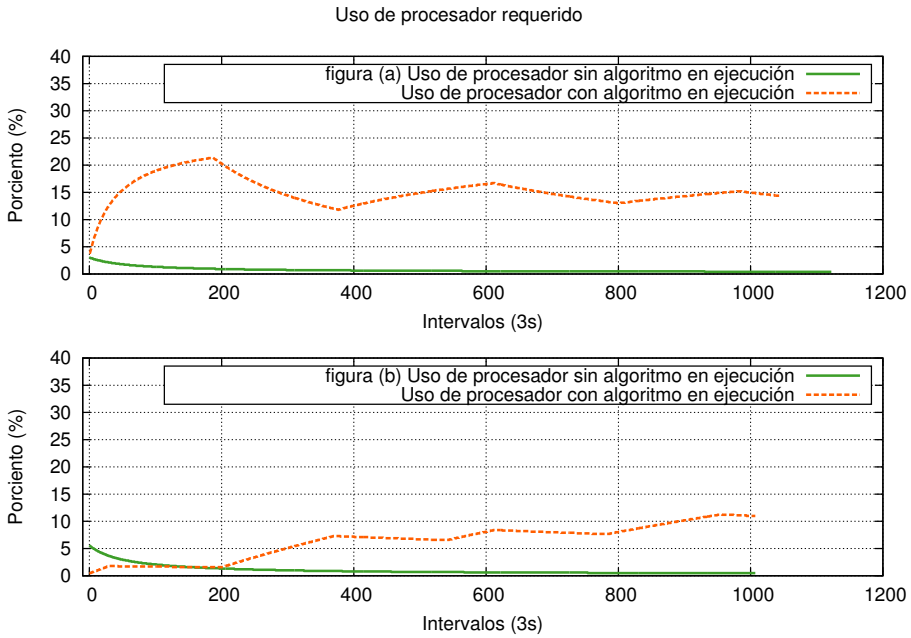

Figura 4.8: Uso de procesador requerido. Ambas figuras muestran el uso del procesador con y sin el algoritmo en ejecución, la figura (a) lo hace para el Equipo 1 y la figura (b) lo hace para el Equipo 2.

El uso del procesador en el *Equipo 1* incrementa hasta el 21.4 % con la primera prueba, al terminar, desciende al 11.8 % y posteriormente la curva se estabiliza alrededor del 15 %. Dado que el 100 % representa los cuatro núcleos del procesador, cada uno corresponde al 25 %, por lo tanto el algoritmo utiliza un solo núcleo alcanzando su máximo de 85.6 % para después descender al 47.2 % y finalmente tiende a estabilizarse en el **60 %**. En el *Equipo 2* el uso del procesador se va incrementando gradualmente hasta alcanzar el 11.3 %, en este caso, el 100 % representa los dos núcleos del procesador, cada uno corresponde al 50 %, por lo tanto las pruebas secuenciales alcanzaron el **22.6 %** de uso en un núcleo del procesador.

Inicialmente el algoritmo SA-SWO fue diseñado para funcionar de manera concurrente

para mejorar su desempeño en equipos con procesadores de múltiples núcleos, sin embargo, el modelo inicial fue modificándose conforme las constantes pruebas efectuadas indicaban que algunas funciones no trabajaban de la manera que se esperaba. Uno de los cambios realizados fue el de utilizar ordenamientos como entrada del SWO, con ello se modificó gran parte de las operaciones para que funcionaran de manera secuencial ya que la concurrencia no respetaba el orden con que se debía operar. Finalmente se redujo el número de funciones concurrentes, dejando solamente algunas actualizaciones, y consecuentemente se lograron mejorar las soluciones que ofrece el algoritmo SA-SWO.

4.2. Caso B: División de Ciencias Básicas

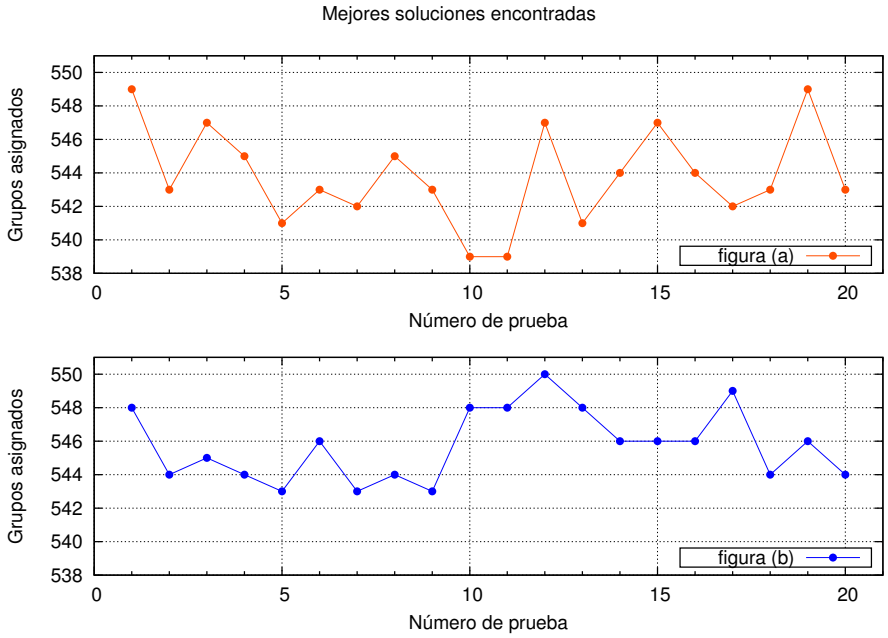

Figura 4.9: Mejores soluciones encontradas. La figura (a) muestra la mejor solución encontrada por prueba en el Equipo 1 y la figura (b) lo muestra para el Equipo 2.

La figura anterior muestra que todas las pruebas realizadas encontraron *soluciones válidas*. El mejor resultado fue el de la prueba 12 en el *Equipo 2* con **550 grupos asignados**, equivalente aproximadamente al **95.65 %** del total de grupos a asignar; los peores resultados

fueron los de las pruebas 10 y 11 en el *Equipo 1* con **539 grupos asignados**, que corresponden aproximadamente al **93.74 %**; y el promedio de las soluciones obtenidas fue de **544 grupos asignados**, aproximadamente el **94.61 %**.

En total, fueron 230 grupos distintos los que no pudieron ser asignados en las pruebas efectuadas, la muestra que emplea la siguiente gráfica toma los 15 primeros con mayor incidencia.

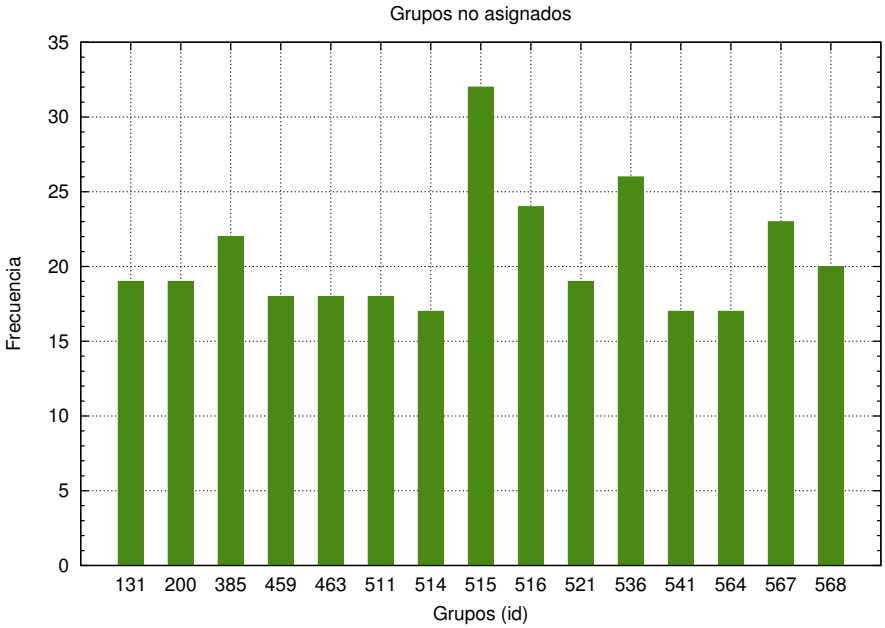

Figura 4.10: Grupos no asignados. La figura muestra los 15 grupos no asignados con mayor frecuencia.

El grupo con *id* 515 fue asignado solamente en ocho de 40 pruebas. En base a la información presentada para la asignación del semestre 2011-1, este *id* pertenece al grupo seis de la asignatura 4314: el Laboratorio de Principios de Termodinámica y Electromagnetismo, clase del profesor S. H. L. A. El horario de disponibilidad del profesor es los días martes de 07:30 a 09:30 horas, tal disponibilidad es exclusiva para esta clase ya que es el único grupo del profesor además de que solamente puede impartirse en el aula H002. La alta frecuencia de este grupo en la gráfica, se debe a que tiene un horario y aula único, además de que la hora de inicio es a las 07:30 horas lo que indica que el grupo puede ser asignado hasta que la

bandera de DLA se encuentra deshabilitada. Por ejemplo, si el grupo aparece como el primer elemento de la permutación X de entrada para el algoritmo SWO, este no será asignado ya que el horario no puede "agregarse" al horario disponible del aula, siendo las 07:00 horas el horario más cercano, lo que indica que el grupo será mandado a la cola del ordenamiento y podrá volverse a tratar de asignar una vez que todos los demás grupos han sido o han tratado de asignarse una vez, ahora con la bandera del DLA deshabilitada.

El aumento de frecuencia en ciertos grupos se debe a su limitado horario disponible y/o aulas que cuenta como válidas para ser asignado de acuerdo a la asignatura.

El Cuadro 4.2 consta de un muestreo de 10 grupos, donde se presentan sus respectivas asignaciones administrativas, *2011-1*, y 3 soluciones generadas por el algoritmo SA-SWO para cada caso: la primera, *SA-SWO 1*, es la solución de la prueba 12 en el *Equipo 2* con 550 grupos asignados, la mejor solución encontrada; la segunda, *SA-SWO 2*, es la solución de la prueba 14 en el *Equipo 2* con 544 grupos asignados, una solución promedio; y la tercera *SA-SWO 3*, es la solución de la prueba 10 en el *Equipo 1* con 539 grupos asignados, una de las peores encontradas.

Grupo (*id*)	Asignación	Horario	Aula (*id2*)
21	2011-1	Martes y Jueves de 07:00-09:15	I103
	SA-SWO 1	Lunes, Miércoles y Jueves de 08:20-09:50	A305
	SA-SWO 2	Martes y Jueves de 07:00-09:15	J109
	SA-SWO 3	Martes y Jueves de 07:00-09:15	J101
131	2011-1	Martes y Jueves de 07:00-09:15	B306
	SA-SWO 1	No Asignado	
	SA-SWO 2	Martes y Jueves de 07:00-09:15	I303
	SA-SWO 3	No Asignado	
330	2011-1	Lunes, Miércoles y Viernes de 19:00-20:30	J109
	SA-SWO 1	Lunes, Miércoles y Viernes de 19:00-20:30	I204
	SA-SWO 2	Lunes, Miércoles y Viernes, 19:00-20:30	I202
	SA-SWO 3	Lunes, Miércoles y Viernes, 10:00-11:30	J108
19	2011-1	Lunes, Miércoles y Viernes de 07:00-08:30	I103
	SA-SWO 1	No Asignado	
	SA-SWO 2	Lunes, Miércoles y Viernes de 08:20-09:50	A305
	SA-SWO 3	Lunes, Miércoles y Viernes de 08:20-09:50	J109
256	2011-1	Martes y Jueves de 09:15-11:30	J101
	SA-SWO 1	Martes y Jueves de 09:15-11:30	I105
	SA-SWO 2	Martes y Jueves, 09:15-11:30	J105
	SA-SWO 3	Martes y Jueves, 09:15-11:30	J102
272	2011-1	Martes y Jueves de 15:30-17:45	I104
	SA-SWO 1	Martes y Jueves de 17:45-20:00	J207
	SA-SWO 2	Martes y Jueves de 17:45-20:00	J207
	SA-SWO 3	Lunes, Miércoles y Viernes de 19:00-20:30	J207
494	2011-1	Martes y Jueves de 17:45-20:00	A201
	SA-SWO 1	Lunes, Miércoles y Viernes de 19:00-20:30	J209
	SA-SWO 2	Lunes, Miércoles y Viernes de 19:00-20:30	J209
	SA-SWO 3	Lunes, Miércoles y Viernes de 19:00-20:30	I304
543	2011-1	Lunes, Miércoles y Viernes de 08:30-10:00	J209
	SA-SWO 1	Lunes, Miércoles y Viernes de 08:30-10:00	I105
	SA-SWO 2	Lunes, Miércoles y Viernes de 08:30-10:00	J102
	SA-SWO 3	Lunes, Miércoles y Viernes de 08:30-10:00	J102
372	2011-1	Martes y Jueves de 17:45-20:00	I106
	SA-SWO 1	Martes y Jueves de 17:45-20:00	J103
	SA-SWO 2	Martes y Jueves de 16:00-18:15	J211
	SA-SWO 3	Lunes, Miércoles y Viernes de 19:00-20:30	J207
12	2011-1	Martes y Jueves de 13:15-15:30	J203
	SA-SWO 1	Martes y Jueves de 13:45-16:00	J203
	SA-SWO 2	Martes y Jueves de 13:15-15:30	J204
	SA-SWO 3	Martes y Jueves de 13:45-16:00	J204

Cuadro 4.2: Caso B. Muestreo de asignaciones.

En el cuadro anterior se visualizan varios aspectos interesantes:

- el grupo con *id* 131 no fue asignado en dos soluciones encontradas por el algoritmo SA-SWO. El conflicto se debe a lo siguiente: este es el grupo número dos de la asignatura

Termodinámica, corresponde al profesor L.P.R, quien además imparte el laboratorio de la misma asignatura, correspondiente al grupo con *id* 151. Su disponibilidad es los martes de 07:00 a 11:15 horas y los jueves de 07:00 a 09:15 horas. Idealmente, solamente hay una manera de que algoritmo pueda asignar los dos grupos: al grupo con *id* 131 debe dar el horario de martes y jueves de 07:00 a 09:15 horas y al grupo con *id* 151 debe asignarlo los martes de 09:15 a 11:15 horas. Cuando no se asigna el grupo con *id* 131, es debido a que el grupo con *id* 151 fue asignado los martes o los jueves de 07:00 a 09:00 horas, con tales asignaciones, no es posible crear para el grupo con *id* 131 un horario de dos sesiones a la misma hora en diferente día, dado que es una condición para que el algoritmo lo construya. En resumen, el grupo no es asignado cuando la distribución de la disponibilidad del profesor no permite crear horarios con esquemas válidos. Esta es una de las razones por las que el algoritmo SA-SWO no puede asignar algunos grupos,

- el grupo con *id* 494 presenta todas las soluciones generadas automáticamente iguales y estas son diferentes a la del semestre 2011-1. Este *id* pertenece al grupo 29 de Ecuaciones Diferenciales y al profesor G. A. R. A., quien además imparte el grupo 17 de Principios de Termodinámica con *id* 513. Su disponibilidad son los días lunes, miércoles y viernes de 19:00 a 20:30 horas y los martes y jueves de 17:45 a 20:00 horas. La selección de ese horario por parte del algoritmo SA se debe a que siempre dará preferencia a los esquemas con mayor número de sesiones además de que el grupo con *id* 494 se encontraba antes que el otro grupo con *id* 513 en el ordenamiento o estaban en orden contrario y este último fue mandado a la cola del mismo,

- el grupo con *id* 543 presenta todas las asignaciones idénticas en horario (caso único del muestro) pero diferentes en la aula. El aula con *id2* J209 es una de las más demandadas en la División, 267 de 575 grupos son candidatos a asignarse en dicha aula. De esos

grupos candidatos los primeros en el ordenamiento serán los que se asignen en ella, probablemente el grupo con *id* 543 no se encontraba en las primeras posiciones de los ordenamientos,

- el único caso donde el algoritmo SA-SWO asignó en la misma aula que en la asignación 2011-1, fue la solución *SA-SWO 1* para el grupo con *id* 12, la diferencia entre ambos resultados es de 30 minutos en el horario. La igualdad en las aulas se debe a que la asignatura del grupo, con *id* 0, solamente puede impartirse en dos de ellas: las aulas con *id2* J203 y J204, y

- del muestreo presentado, ninguna asignación generada por el algoritmo SA-SWO es idéntica a la presentada en la asignación del semestre 2011-1.

Por otro lado, las gráficas de la Figura 4.11 presentan los tiempos de ejecución de cada prueba en sus respectivos equipos.

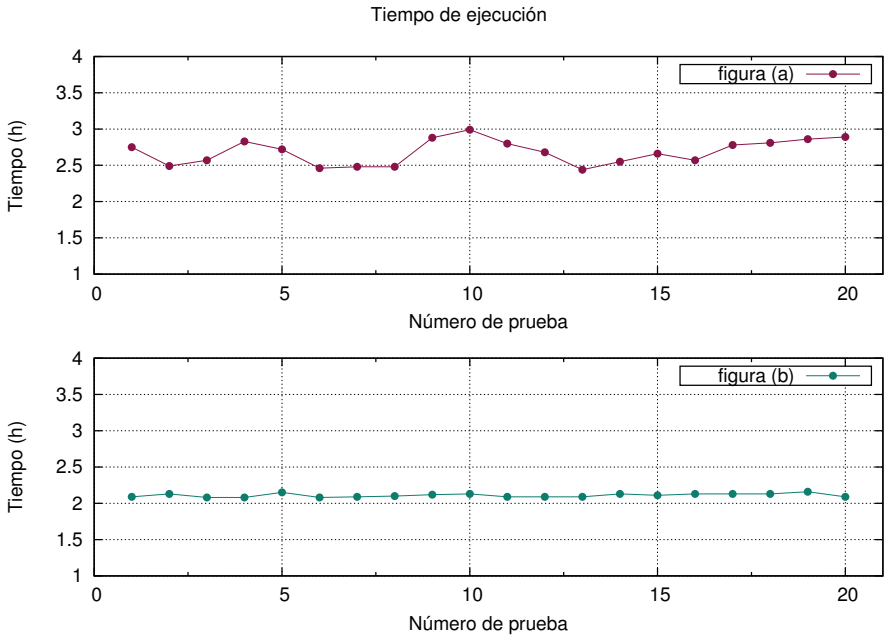

Figura 4.11: Tiempo de ejecución. La figura (a) muestra los tiempos de ejecución por prueba en el Equipo 1; la figura (b) lo muestra para el Equipo 2.

El tiempo promedio para el *Equipo 1* fue de **2.68 horas** y para el *Equipo 2* fue de **2.11 horas**. La diferencia entre ambos promedios fue de **0.57 horas**. En este caso, el tiempo requerido para alcanzar la temperatura final es el mismo valor que el tiempo de ejecución, ya que todas las soluciones encontradas fueron *soluciones válidas*.

Con la información obtenida, no es posible calcular el tiempo promedio que tarda el algoritmo en encontrar *soluciones factibles*, debido a que no se pudo obtener ese tipo de respuestas. Cabe mencionar que el *Caso B* tiene al menos una *solución factible* y es formada por un subconjunto de la asignación de División de Ciencias Básicas presentada en el semestre 2011-1, en este caso, el hecho de no encontrar alguna *solución factible* no quiere decir que ésta no exista o que el algoritmo no funciona.

Las siguientes gráficas muestran el consumo de memoria y procesador. Las pruebas fueron ejecutadas de manera secuencial separadas por intervalos de diez minutos en un lapso de seis horas.

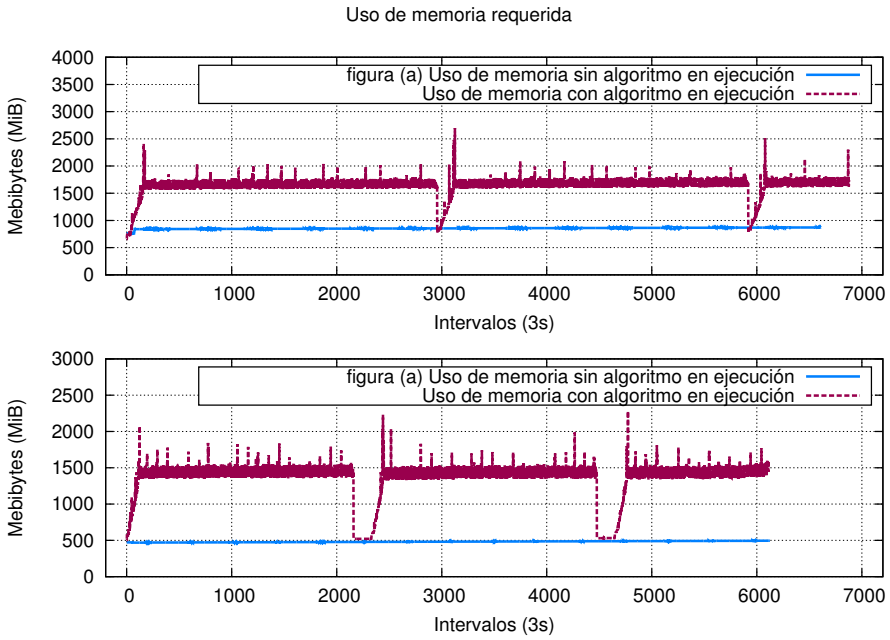

Figura 4.12: Uso de memoria requerida. La figura (a) muestra el consumo de memoria en el Equipo 1 mientras que la figura (b) lo muestra para el Equipo 2.

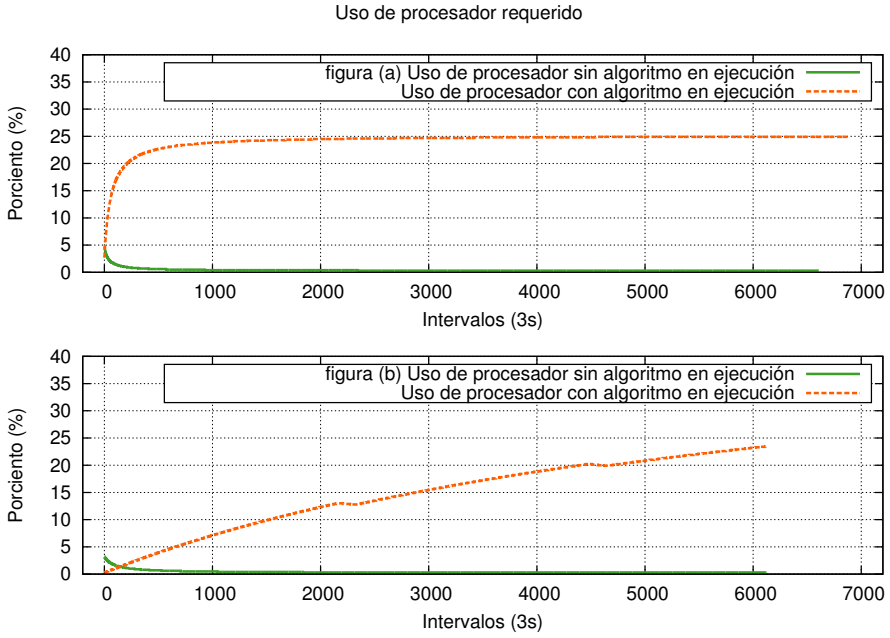

Figura 4.13: Uso de procesador requerido. Las gráficas muestran el uso del procesador con y sin el algoritmo en ejecución, la figura (a) lo hace para el Equipo 1 y la figura (b) lo hace para el Equipo 2.

Equipo	Consumo "normal" (MiB)	Consumo con algoritmo en ejecución(MiB)	Memoria requerida (MiB)
1	854.95	1649.49	794.54
2	481.48	1449.28	967.80

Cuadro 4.3: Caso B. Promedios de consumo de memoria. Datos duros obtenidos de la información presentada en las gráficas de la Figura 4.12.

El uso de procesador en la figura (a) rápidamente llega al 24.9 % y se mantiene en ese valor. Por otro lado, la figura (b) muestra un incremento casi lineal hasta llegar al 23.5 %, un consumo poco menor de la mitad de un núcleo del procesador, lo cual es relativamente bajo comparado con el del *Equipo 1* que utiliza cerca de la totalidad de uno de los cuatro núcleos que posee.

Este caso en general es más restrictivo en comparación con el *Caso A*, ya que la disponibilidad de los profesores es exacta al total de número de grupos que debe impartir. De igual manera las aulas fueron delimitadas de tal manera que para toda asignatura su conjunto de aulas disponibles esta formado por todas aquellas que tuvieran al menos un grupo asignado con dicha asignatura.

Para alcanzar con mayor facilidad *soluciones factibles* para este caso, se requiere modificar algunos datos de entrada que permitan mayor libertad en las disponibilidades y las aulas compatibles para cada asignatura. Hacerlo sin el consentimiento de los profesores y/o alguna autoridad que responda y valide la nueva información modelaría un caso distinto probablemente lejano a la realidad.

4.3. Caso C: hdtt4-8

El Cuadro 4.4 muestra los resultados obtenidos en diferentes algoritmos utilizando los mismos problemas que conforman el *Caso C*.

Algoritmo	hdtt4		hdtt5		hdtt6		hdtt7		hdtt8	
	RF1	RF2	RF1	RF2	RF1	RF2	RF1	RF2	RF1	RF2
GS	5	8.5	11	16.2	19	22.2	26	30.9	29	35.4
NN-TT2	0	0.1	0	0.5	0	0.8	0	1.1	0	1.4
NN-TT3	0	0.5	0	0.5	0	0.7	0	1	0	1.2
SA1	-	-	0	0.7	0	2.5	2	2.5	2	2.5
SA2	**0**	**0**	0	0.3	0	0.8	0	1.2	0	1.9
TS	0	0.2	0	2.2	3	5.6	4	10.9	13	17.2
CPMF	5	10.7	8	13.2	11	18.7	18	25.6	15	28.6
DWTAN	**0**	**0**	0	0.4	0	1.65	0	2.1	0	3.25
SA3	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0	0.4
EAH	**0**	**0**	2	5.4	6	7.9	9	12	13	15.1
TFH	**0**	**0**	0	0.6	0	2.1	0	2.5	0	3.1
SA-SWO	**0**	**0**	0	0.4	0	1.95	3	3.95	3	4.9

Cuadro 4.4: Caso C. Resultados de los problemas hdtt empleando distintos algoritmos. Los datos fueron tomados de [Pim13].

La primera columna del cuadro anterior indica los algoritmos utilizados: GS (Greedy Search), NN-TTx (Neural Networks), SAx y TS pueden consultarse en [Smi03][68]; CPMF (Continuous Potts Mean-Field annealing approach) y DWTAN (Discrete Winner-Take-All Neuron approach) en [Car04]; SA3 en [Liu09] y EAH (Event-Assignment Heuristic) y TFH (Timeslot-Filling Heuristic) en [Pim13]. Las columnas siguientes indican el nombre de cada uno de los problemas, donde cada uno tiene dos resultados: RF1 y RF2. El primero representa el número de restricciones fuertes quebrantadas en la mejor solución encontrada en cada caso y el segundo representa el promedio de restricciones quebrantadas en 20 pruebas ejecutadas para cada problema.[69]

Para los problemas hdtt4, hdtt5 y hdtt6, el algoritmo SA-SWO logró encontrar *soluciones factibles* para cada caso, de hecho, para hdtt4 las 20 pruebas ejecutadas regresaron el mismo tipo de solución. Para los casos hdtt7 y hdtt8 las mejores soluciones fueron *soluciones válidas* y cada una dejó tres grupos sin asignar.

Comparando los datos del cuadro 4.4, para el caso hdtt4, el SA-SWO obtiene la misma

[68]La "x" denota los números que diferencian los tipos de algoritmos.

[69]Para el algoritmo SA1, el número de pruebas ejecutadas fueron 6 en cada caso. Para SA-SWO, se seleccionaron de manera pseudo-aleatoria 20 pruebas de las 40 ejecutadas además de que las "restricciones fuertes quebrantadas" son el número de grupos no asignados en cada caso, dado que siempre regresa soluciones que respetan todas las restricciones especificadas.

calidad de soluciones que los algoritmos SA2, DWTAN, SA3, EAH y TFH; para los demás casos, el SA-SWO no siempre pudo regresar *soluciones factibles*, sin embargo sus valores de RF2 son menores que los obtenidos por otros algoritmos, por ejemplo, en hdtt5 es menor que el obtenido por TS pero ligeramente mayor que el de SA2; en hdtt7 es mayor que los valores de NN-TT2 y NN-TT3 pero menor que el obtenido en GS. Cabe mencionar que el algoritmo SA3, obtuvo los mejores resultados del estudio, ya que a excepción del problema hdtt8, sus soluciones no rompen ninguna *restricción fuerte*.

En este tipo de problemas, donde inicialmente todos los profesores comparten una cantidad común de grupos (30 en este caso) y de *ventanas de tiempo disponibles*, todos los grupos tienen las mismas posibilidades de no ser asignados, la frecuencia de ellos dependerá de los ordenamientos y las probabilidades generadas por el SA-SWO.

A continuación se muestran dos tablas con los datos obtenidos durante las pruebas realizadas referentes al tiempo de ejecución y consumo de recursos en los equipos de cómputo utilizados.

Problema	**Equipo 1**			**Equipo 2**		
	t_{best}	$\overline{t_{sf}}$	$\overline{t_{sv}}$	t_{best}	$\overline{t_{sf}}$	$\overline{t_{sv}}$
hdtt4	2.26	24.40	-	1.83	24.33	78.88
hdtt5	15.71	80.51	192.73	1.97	97.43	185.66
hdtt6	241.35	241.35	578.84	83.41	222.32	567.30
hdtt7	-	-	558.64	-	-	546.42
hdtt8	-	-	760.50	-	-	686.85

Cuadro 4.5: Caso C. Tiempos de ejecución. El cuadro muestra los promedios de los tiempos de ejecución obtenidos de las pruebas realizadas. t_{best} indica el mejor tiempo conseguido por una *solución factible*; $\overline{t_{sf}}$ indica el tiempo promedio generado por las *soluciones factibles* y $\overline{t_{sv}}$ indica el tiempo promedio generado por las *soluciones válidas*. Los valores están en segundos.

Equipo	**hdtt4**		**hdtt5**		**hdtt6**		**hdtt7**		**hdtt8**	
	$\overline{M}$	$\overline{P}$	$\overline{M}$	$\overline{P}$	$\overline{M}$	$\overline{P}$	$\overline{M}$	$\overline{P}$	$\overline{M}$	$\overline{P}$
1	7.23	4.8	21.75	18.4	35.63	30.6	49.21	44.8	63.11	56.2
2	10.36	1.5	26.12	7.25	39.78	14.04	52.49	22.38	66.35	25.61

Cuadro 4.6: Caso C. Cantidad de memoria y procesador requeridos. En este cuadro, $\overline{M}$ representa el promedio de memoria requerida en (MiB) y $\overline{P}$ representa el promedio del uso de procesador (en % y sobre un núcleo), que demanda el algoritmo SA-SWO en ejecución. Al igual que en el *Caso A*, el tiempo de muestreo en que se tomaron los datos fue de una hora.

4.4. Análisis del algoritmo SA-SWO

De acuerdo a los resultados obtenidos, el *Equipo 1* mostró requerir menor cantidad de memoria que el *Equipo 2*, sin embargo su uso de procesador converge más rápidamente a ocupar por completo un núcleo y se mantiene constante en ese punto, a diferencia del uso que tiene el procesador del *Equipo 2*, el cual se incrementa de manera más uniforme conforme al tiempo sin llegar a ocupar todo un núcleo. Por otro lado, los tiempos de ejecución de las pruebas en el *Equipo 2* mostraron ser más uniformes entre si y menores que los obtenidos en el *Equipo 1*. Puede concluirse que el desempeño del algoritmo SA-SWO en el *Equipo 2* fue mejor gracias a la frecuencia con que trabaja su procesador, un dato importante al momento de considerar otros equipos para realizar pruebas futuras.

El mejor caso en tiempo de ejecución puede considerarse cuando la primera Θ generada tenga energía igual a cero, es decir, que todos los grupos sean asignados a partir de la primera permutación aleatoria X y tal permutación debe ser igual a la secuencia ordenada de grupos asignados Y, lo que significa que todos los grupos fueron asignados en el primer intento, la bandera DLA se mantuvo habilitada y por lo tanto la parte del SWO nunca entró al bloque *Agrupador*. Este escenario en particular es muy difícil de obtener y dependerá del número de permutaciones existentes que cumplan con la condición anterior. Por otro lado, el peor caso en tiempo de ejecución se considera cuando no se encuentra una solución factible y el algoritmo alcanza el parámetro de paro: la *Temperatura final*.

Para el análisis asintótico del algoritmo SA-SWO usando la $notación - O$, se requiere una entrada n que represente el número de grupos que se busca asignar. En general, las bloques de condiciones, ciclos y asignaciones se consideran términos constantes y se representan como $O(1)$, por lo que solamente se toman en cuenta aquellas funciones que operan con los elementos de n. En la función SWO (Algoritmo 4), el $Analizador$ opera con cada elemento de n y su complejidad se representa como $O(n)$ y el $Constructor$ opera con un solo elemento

de n así que puede representarse, como $O(1)$, como ambas funciones se encuentran dentro de un ciclo que opera con todos los elementos de n, la complejidad de la función SWO es $O(n^2)$. Para la *Función conflicto* (Algoritmo 2) su complejidad se representa como $O(n^2)$ dado que en dos ciclos anidados se opera sobre todos los elementos de n. Finalmente, la complejidad del algoritmo SA-SWO es $\mathbf{O(n^2)}$.

De acuerdo a la **Definición 7**, el algoritmo SA-SWO mostró ser **incompleto** ya que siempre regresa una solución y ésta puede no incluir a todos los grupos asignados. Al ser incompleto, garantiza que al menos siempre se obtendrá una *solución válida* ya que el bloque *Constructor* no permite quebrantar las restricciones fuertes (2.3), (2.4) y (2.5). El único caso donde se puede asegurar que el problema tiene alguna *solución factible* es cuando la cardinalidad de Θ, el conjunto de asignaciones, es igual a la cardinalidad de G, el conjunto de los grupos, es decir, cuando $|\Theta| = |G|$. Al calcular la *Tabla de conflictos* solamente se consideran aquellos grupos que pueden tener un horario idéntico entre ellos, los demás grupos con posibles horarios que solamente pueden tomar una parte de éste no son almacenados dado el alto costo computacional que involucran, por ello, este criterio puede no permitir alcanzar el óptimo y consecuentemente no lograr una *solución factible*.

En algunos casos, explorar completamente el espacio de búsqueda puede ser sumamente costoso respecto al tiempo y no hay garantía de hacerlo siempre, por ello se requiere analizar los resultados para buscar el equilibrio entre tiempo y calidad de respuesta y de ser necesario, modificar los resultados con *asignaciones especiales* que permitan mejorar la solución actual.

Capítulo 5

Conclusiones y trabajo futuro

El algoritmo SA-SWO demostró un buen funcionamiento ofreciendo soluciones aceptables en tiempo razonable para los casos de estudio. Un beneficio se vería en la reducción considerable del tiempo requerido para resolver el caso de División de Ciencias Básicas empleando el algoritmo presentado. Otra ventaja de usarlo en un caso real de la FI es el hecho de no depender de las asignaciones de semestres anteriores las cuales no siempre fueron *soluciones factibles*.

Sin embargo, existen diversos criterios adicionales que no se tomaron en cuenta y que podrían ayudar a conseguir diferentes soluciones. Algunas rutas propuestas a considerar para trabajo futuro son:

- probar el algoritmo con los datos de alguna otra división, por ejemplo la de Ciencias Sociales y Humanidades donde la mayoría de sus grupos presentan una distribución uniforme en tanto duración de las sesiones y aulas ocupadas,
- correr pruebas incrementando gradualmente el número de grupos hasta llegar al total, eliminando los casos especiales que requieren asignarse manualmente[70] y adaptando

[70]Descritos en la Sección *3.5 Alcance y limitaciones*.

los parámetros del algoritmo,[71]

- incluir la preferencia de horarios y aulas para los grupos de los profesores dentro de la entrada del algoritmo para que éstos sean los primeros que traten de asignarse, considerándolos como restricciones débiles, podrían conducir a soluciones satisfactorias más rápidamente,

- agregar módulos de "Optimización" donde se contemplen restricciones débiles que agreguen valor adicional a las asignaciones, por ejemplo, se podría buscar reducir intersecciones en los grupos de asignaturas seriadas y pertenecientes al mismo semestre, con el fin de facilitar la creación de horarios para los alumnos; otra restricción podría ser asignar los grupos a los profesores en aulas cercanas cuando éste tenga sesiones con horario continuo y aulas más alejadas cuando tenga minutos libres entre clases, así tendría tiempo suficiente para trasladarse y no restar tiempo a las sesiones,

- buscar nuevos tipos de ordenamientos para que presenten mejores resultados o permitan explorar de diferente manera el espacio de búsqueda, y

- añadir más restricciones que permitan modelar los archivos de entrada lo más cercano posible a la realidad. Por ejemplo, una restricción fuerte consistiría en agregar la capacidad del aula, así se contemplaría para los grupos de alta demanda aulas con capacidad suficiente, como los grupos de Álgebra y Cálculo Diferencial, dejando las aulas de menor capacidad para grupos con poca demanda, como los grupos del módulo terminal de Computación Gráfica y Sistemas Inteligentes; por otro lado, una restricción débil podría considerar asignar grupos en horarios nocturnos cerca de las escaleras y/o en la planta baja y primer piso, lo cual sería aconsejable pero no necesario.

[71] Los parámetros *Temperatura final*, *Temperatura inicial*, *Constante de normalización*, el *Número de iteraciones* y la *Constante de enfriamiento* inicialmente deben proponerse de manera arbitraria para posteriormente ser establecidos después de ejecutar y analizar varias pruebas. Particularmente, la *Constante de enfriamiento* puede calcularse usando funciones de comportamiento exponencial, logarítmico, entre otras.

Es importante mencionar que las pruebas aportan información valiosa para delimitar el problema y mejorar los métodos utilizados para la asignación de grupos, en algunos casos cualquier respuesta es mejor a no tener ninguna.

La investigación queda abierta, la estrategia presentada para ofrecer soluciones al problema de *timetabling* que enfrenta la FI es una alternativa para encontrar diferentes soluciones. Trabajar otros algoritmos de Inteligencia Artificial y estar al tanto del estado del arte en esta disciplina, puede en su momento y debido trabajo, ofrecer distintas soluciones que se adapten de la mejor manera posible a los requerimientos por parte de la facultad.

Apéndice A

Solución factible base para el Caso A

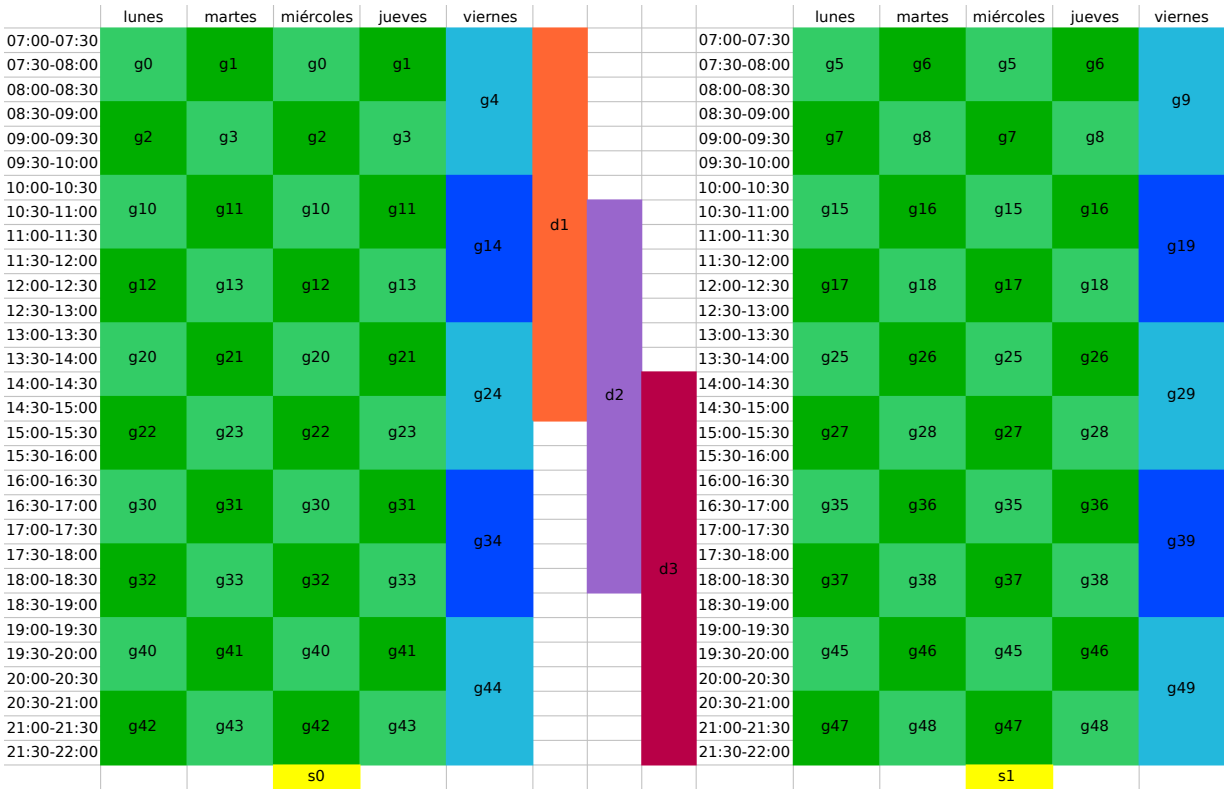

Figura A.1: Solución factible base para el Caso A.

Esta representación gráfica de la solución base muestra las siguientes características:

- Las aulas $s0$ y $s1$ (indicadas en amarillo) tienen horario disponible de lunes a viernes

de 07:00 a 22:00 horas.

- Los tres tipos de disponibilidades de los profesores se indican como: $d1$ (color anaranjado) de lunes a viernes de 07:00 a 15:00 horas; $d2$ (color morado) de lunes a viernes de 10:30 a 18:30 horas; y $d3$ (color rosa) de lunes a viernes de 14:00 a 22:00 horas.

- Cada aula tiene 25 grupos asignados y se encuentran ocupadas al 100 % de su capacidad.

- El esquema de diez grupos es de una sesión de tres horas, asignados el día viernes, y el esquema de los 40 grupos restantes es de dos sesiones de una hora y media distribuidos de lunes a jueves.

- Los grupos de un mismo profesor son continuos. Por ejemplo, al profesor $p0$ le corresponden los grupos $g0$, $g1$, $g2$, $g3$ y $g4$, juntando sus horarios, el profesor imparte clases de lunes a viernes de 07:00 a 10:00 horas en el aula $s0$. La disponibilidad restante del profesor es de 10:00 a 14:00 horas.

- La solución se presta para intercambiar grupos que presenten el mismo esquema y disponibilidad. Por ejemplo, el grupo $g0$, perteneciente al profesor $p0$, puede intercambiarse con el grupo $g5$, perteneciente al profesor $p1$, gracias a que sus características son similares.

Apéndice B

Recomendaciones para uso del algoritmo SA-SWO

Garantizar que existe un algoritmo que resuelve de manera factible y eficiente todos los casos de un problema de *timetabling*, no ha sido posible, tampoco el que siempre exista al menos una solución que cumpla todos los requerimientos de entrada definidos, sin embargo, se pueden ejecutar varias pruebas utilizando el algoritmo SA-SWO y analizar los resultados para:

- establecer parámetros adecuados para que las soluciones encontradas puedan ser útiles en la práctica,
- buscar si existe alguna *solución factible* o un conjunto de éstas, o en su defecto,
- localizar grupos que por sus disponibilidades no permiten llegar a este tipo de solución.

Es recomendable *relajar*[72] lo más posible los parámetros de entrada para facilitar la búsqueda de soluciones que asignen la mayor cantidad de grupos cumpliendo las restricciones indicadas.

[72]El término *relajar* refiere a volver menos rigurosas las restricciones empleadas, como las disponibilidades de profesores o aulas, por mencionar un ejemplo.

Los horarios de disponibilidad continua de las aulas para todas las asignaturas que puedan impartirse en ellas son un parámetro importante, ya que el algoritmo contaría con espacio para explorar y asignar grupos sin tener la restricción de horarios reservados y de que ciertas aulas pertenezcan a alguna división en particular.

Las peticiones por parte de los profesores para programar sus grupos permite identificar dos tendencias principales que clasifican a los horarios en "alta demanda" y "baja demanda". Los horarios de "alta demanda" se concentran de 11:00 a 13:00 horas, particularmente, los martes y jueves, ya que son considerados "cómodos" tanto para profesores como para estudiantes. Por otra parte, los horarios de "baja demanda" abarcan de 19:00 a 22:15 de lunes a viernes y de 07:00 a 14:30 los sábados.[73]

Las disponibilidades de los profesores de acuerdo a su nombramiento también podrían jugar un papel importante al momento de asignar grupos: aquellos que tengan disponibilidades de 40 o 20 horas a la semana deberían ser especificadas, el espacio de búsqueda puede crecer considerablemente pero permitiría a los grupos de estos profesores ser asignadas en horarios que no sean de "alta demanda" y permitir a aquellos profesores de asignatura o de disponibilidad restringida, asignarles sus grupos en horarios más específicos y muchas veces únicos.

Por último, una recomendación fuera del ámbito computacional, es informar al personal académico sobre la complejidad del problema, los beneficios y límites del algoritmo presentado y la flexibilidad para adoptar una nueva tecnología que lograría un avance favorable en la administración de grupos.

[73]Entrevista con el secretario de servicios académicos de la FI, Lic. Miguel Figueroa Bustos, noviembre 2011.

Glosario

autómata finito

Es un modelo matemático que representa la abstracción de un sistema el cual consta solamente de un número finito de estados (descripciones instantáneas que ofrecen información de cómo el sistema puede evolucionar desde este punto) y sus transiciones alcanzables (cambios de estado que pueden suceder de manera espontánea o en respuesta a entradas externas) [Koz97].

frentes Pareto

Son conjuntos no dominados donde dos individuos $x_1, x_2 \in \mathbb{X}$, no cumplen que $x_1 \vdash x_2$ ni que $x_2 \vdash x_1$ [Wei09].

polinomio

Es una expresión matemática representada por la suma de términos formados por el producto entre variables y constantes [Sip13].

Bibliografía

[Al-09] Al-Betar, M. A. and Khader, A. T. A Hybrid Harmony Search for University Course Timetabling. In Blazewicz, J. and Drozdowski, M. and Kendall, G. and McCollum, B., editor, *Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2009)*, pages 157–179, Dublin, Ireland, 2009. `http://www.mistaconference.org/2009/papers/157-179-113-P.pdf` [Online; accessed 22-June-2011].

[Al-10] Al-Betar, M. A. and Khader, A. T. and Liao, I. Y. A Harmony Search with Multi-pitch Adjusting Rate for the University Course Timetabling. In Geem, Z. W., editor, *Recent Advances In Harmony Search Algorithm*, volume 270 of *Studies in Computational Intelligence*, pages 147–161. Springer Berlin Heidelberg, 2010.

[Arm07] Armstrong, J. *Programming Erlang: Software for a Concurrent World*. Pragmatic Bookshelf, USA, 2007.

[Aro09] Arora, S. and Barak, B. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 2009.

[Bol13] Boltzmann constant. `https://www.princeton.edu/~achaney/tmve/wiki100k/docs/Boltzmann_constant.html` [Online; accessed 30-may-2013], May 2013.

[Bou04] Bourke, P. DLA - Diffusion Limited Aggregation. http://paulbourke.net/fractals/dla/ [Online; accessed 14-June-2013], January 2004.

[Bur04] Burke, E. K. and Newall, J. P. Solving Examination Timetabling Problems through Adaption of Heuristic Orderings. *Annals of Operations Research*, 129(1-4):107–134, 2004. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.62.6081&rep=rep1&type=pdf[Online; accessed 24-June-2011].

[Bur09] Burke, E. K. and Hyde, M. and Kendall, G. and Ochoa, G. and Ozcan, E. and Qu, R. A Survey of Hyper-heuristics. Technical report, School of Computer Science and Information Technology, University of Nottingham, Nottingham NG8 1BB, UK, March 2009. http://www.cs.nott.ac.uk/TR/SUB/SUB-0906241418-2747.pdf [Online; accessed 9-August-2011].

[Car04] Carrasco, M. P. and Pato, M. V. A Comparison of Discrete and Continuous Neural Networks Approaches to Solve the Class/Teacher Timetabling Problem. *European Journal of Operational Research*, 153(1):65–79, 2004. https://www.repository.utl.pt/bitstream/10400.5/1427/1/MVP-4.2001.pdf [Online; accessed 12-August-2013].

[Coo71] Cook, S. A. The Complexity of Theorem-Proving Procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM. http://www.cs.toronto.edu/~sacook/homepage/1971.pdf [Online; accessed 18-February-2011].

[Cop08] Copeland, B. J. The Church-Turing Thesis. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*. Fall 2008 edition, 2008. http://plato.stanford.edu/archives/fall2008/entries/church-turing/ [Online; accessed 27-July-2011].

[Cor09] Cormen, T. H. and Leiserson, C. E. and Rivest, R. L. and Stein, C. *Introduction to Algorithms*. McGraw-Hill Higher Education, USA, third edition, 2009.

[Dat06] Datta, D. and Deb, K. and Fonseca, C. M. Solving Class Timetabling Problem of IIT Kanpur using Multi-Objective Evolutionary Algorithm. `http://www.iitk.ac.in/kangal/papers/k2006006.pdf` [Online; accessed 22-June-2011], June 2006.

[Deb00] Deb, K. and Agrawal, S. and Pratap, A. and Meyarivan, T. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. In *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, PPSN VI, pages 849–858, London, UK, UK, 2000. Springer-Verlag. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.18.4257&rep=rep1&type=pdf` [Online; accessed 18-February-2011].

[Dor04] Dorigo, M. and Stützle, T. *Ant Colony Optimization*. A Bradford Book, USA, 2004.

[Eve76] Even, S. and Iati, A. and Shamir, A. On the Complexity of Timetable and Multicommodity Flow Problems. *SIAM Journal of Computation 5*, (4):691–703, 1976. `http://www.cs.technion.ac.il/~itai/publications/Algorithms/multi-commodity-timetables.pdf` [Online; accessed 16-October-2011].

[Fen05] Feng, G. and Lau, H.C. Efficient Algorithms for Machine Scheduling Problems with Earliness and Tardiness Penalties. In Kendall, G. and Lei, L. and Pinedo, M., editor, *Proceedings of the 1st Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA)*, volume I, pages 196–212, New York, USA, 2005. `http://www.mistaconference.org/2005/papers/Efficient%20Algorithms%20for%20Machine%20Scheduling%20Problems%`

20with%20Earliness%20and%20Tardiness%20Penalties.pdf[Online; accessed 24-June-2011].

[Fra07] Frausto-Solís, J. and Alonso-Pecina, F. and Gonzalez-Segura, C. Analytically Tuned Parameters of Simulated Annealing for the Timetabling Problem. In *Proceedings of the 6th WSEAS International Conference on Computational Intelligence, Man-Machine Systems and Cybernetics*, CIMMACS '07, pages 17–22, Stevens Point, Wisconsin, USA, 2007. World Scientific and Engineering Academy and Society (WSEAS). http://www.cinhtia.com.mx/uady/publicaciones/25-707.pdf [Online; accessed 25-June-2011].

[Gar79] Garey, M. R. and Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, NY, USA, 1979.

[Gar11] García-Villoria, A. and Salhi, S. and Corominas, A. and Pastor, R. Hyper-heuristic Approaches for the Response Time Variability Problem. *European Journal of Operational Research*, 211(1):160–169, May 2011. http://ideas.repec.org/a/eee/ejores/v211y2011i1p160-169.html [Online; accessed 10-August-2011].

[Gas01] Gaspero, L. D. and Schaerf, A. Tabu Search Techniques for Examination Timetabling. In *Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III*, PATAT '00, pages 104–117, London, UK, UK, 2001. Springer-Verlag. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.107.3191&rep=rep1&type=pdf[Online; accessed 21-June-2011].

[Gas07] Gaspero, L. D. and McCollum, B. and Schaerf, A. The Second International Timetabling Competition (ITC-2007): Curriculum-based Course Timetabling (Track 3). In Pecora, F. and Policella, N., editor, *Proceedings of the 1st Inter-*

national Workshop on Scheduling a Scheduling Competition (SSC 2007), Providence (RI), USA, September 2007. http://pst.istc.cnr.it/RCRA07/articoli/P08-digaspero-etal-RCRA07.pdf [Online; accessed 4-September-2011].

[Gem10] Gemm, Z. W. State-of-the-Art in the Structure of Harmony Search Algorithm. In *Recent Advances In Harmony Search Algorithm*, pages 1–10. 2010. http://johnshopkins.academia.edu/ZongWooGeem/Papers/450693/State-of-theArt_in_the_Structure_of_Harmony_Search_Algorithm [Online; accessed 10-February-2011].

[Glo97] Glover, F. and Laguna, M. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[Gue11] Guerrero Z., J. G. Informe de Actividades 2010. http://www.ingenieria.unam.mx/informe2010/informe_2010.pdf [En línea; acceso 15-Agosto-2011], Febrero 2011.

[Hop01] Hopcroft, J. E. and Motwani, R. and Ullman, J. D. *Introduction to Automata Theory, Languages, and Computation*. Pearson Education, USA, Second edition, 2001.

[Jat11] Jat, S. and Yang, S. A Guided Search Non-dominated Sorting Genetic Algorithm for the Multi-Objective University Course Timetabling Problem. In *Proceedings of the 11th European Conference on Evolutionary Computation in Combinatorial Optimization*, EvoCOP'11, pages 1–13, Berlin, Heidelberg, 2011. Springer-Verlag. http://dspace.brunel.ac.uk/bitstream/2438/5983/4/Fulltext.pdf [Online; accessed 22-June-2011].

[Kar72] Karp, R. M. Reducibility Among Combinatorial Problems. In Miller R. E. and Thatcher J. W., editor, *Complexity of Computer Computations*, pages 85–103, New York, USA, 1972. Plenum Press.

[Knu97] Knuth, D. E. *The Art of Computer Programming. Volume 1: Fundamental Algorithms*. Addison-Wesley, 1997.

[Koz97] Kozen, D. C. *Automata and Computability*. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[Lew07] Lewis, R. and Paechter, B. and McCollum, B. Post Enrolment based Course Timetabling: A Description of the Problem Model used for Track Two of the Second International Timetabling Competition. Technical report, Cardiff University, Wales, UK, August 2007. `http://business.cardiff.ac.uk/sites/default/files/A2007_3%20%281%29.pdf`[Online; accessed 10-June-2011].

[Liu09] Liu, Y. and Zhang, D. and Leung, S.C.H. A Simulated Annealing Algorithm with a New Neighborhood Structure for the Timetabling Problem. In *Proceedings of the First ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, GEC '09, pages 381–386, New York, NY, USA, 2009. ACM. `http://59.77.16.8/DownLoad/A%20Simulated%20Annealing%20with%20a%20New%20Neighborhood%20Structure%20Based%20Algorithm%20for%20High%20School%20Timetabling%20Problems.pdf` [Online; accessed 12-August-2013].

[Mea98] Meakin, P. *Fractals, Scaling and Growth Far From Equilibrium*. Cambridge University Press, Cambridge, United Kingdom, 1998.

[Mit98] Mitchell, M. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1998.

[Pea84] Pearl, J. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Publishing Company, Boston, MA, USA, 1984.

[Per06] Perzina, R. Solving the University Timetabling Problem with Optimized Enrollment of Students by a Self-adaptive Genetic Algorithm. In Burke, E. K. and Rudová, H., editor, *Practice and Theory of Automated Timetabling VI*, volume VI of *Lecture Notes in Computer Science*, pages 248–263, Brno, Czech Republic, August/September 2006. 6th International Conference, PATAT 2006, Springer.

[Pet04] Petrovic, S. and Patel, V. and Yang, Y. Examination Timetabling with Fuzzy Constraints. In Burke, E. and Trick, M., editor, *Practice and Theory of Automated Timetabling V*, volume V of *Lecture Notes in Computer Science*, pages 313–233, Pittsburgh, PA, USA, August 2004. 5th International Conference, PATAT 2004, Springer.

[Pim13] Pimmer, M. and Raidl, G. A Timeslot-Filling Heuristic Approach to Construct High-School Timetables. In Di Gaspero, L. and Schaerf, A. and Stützle, T., editor, *Advances in Metaheuristics*, volume 53 of *Operations Research/Computer Science Interfaces Series*, pages 143–157. Springer New York, 2013. `https://www.ads.tuwien.ac.at/publications/bib/pdf/pimmer-12.pdf` [Online; accessed 12-August-2013].

[Qu,09] Qu, R. and Burke, E. K. and Mccollum, B. and Merlot, L. T. and Lee, S. Y. A Survey of Search Methodologies and Automated System Development for Examination Timetabling. *J. of Scheduling*, 12(4):55–89, February 2009. `http://red.cs.nott.ac.uk/~rxq/files/Exam_Review.pdf` [Online; accessed 22-June-2011].

[Ran01] Randall, M. and Abramson, D. A General Meta-Heuristic Based Solver for Combinatorial Optimization Problems. *Computational Optimization and Applications*, 20(2):185–210, 2001. http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=BD7108259B794465944ABAC36AD000DE?doi=10.1.1.43.6225&rep=rep1&type=%pdf [Online; accessed 12-August-2013].

[Rus03] Russell, S. and Norving, P. *Artificial Intelligence A Modern Approach*. Prentice Hall, USA, Second edition, 2003.

[Sch11] Schmidt, N. Simulated Annealing. http://drc.denison.edu/bitstream/handle/2374.DEN/5099/schmidt.pdf?sequence=1 [Online; accessed 15-February-2011], February 2011.

[Sip13] Sipser, M. *Introduction to the Theory of Computation*. Cengage Learning, Boston, USA, Third edition, 2013.

[Smi99] Smith, K.A. and Abramson, D. and Duke, D. Efficient Timetabling Formulations for Hopfield Neural Networks. In *Artificial Neural Networks in Engineering Conference (ANNIE'99)*, 1999. http://messagelab.monash.edu.au/Publications/Publication?action=download&upname=effic_tt_hopfield.pdf [Online; accessed 12-August-2013].

[Smi03] Smith, K. A. and Abramson, D. and Duke, D. Hopfield Neural Networks for Timetabling: Formulations, Methods, and Comparative Results. *Comput. Ind. Eng.*, 44(2):283–305, February 2003. http://ocw.up.edu.ps/repositories/server_america/file_rar/genetic_research/genetic_research/cie2003.pdf [Online; accessed 12-August-2013].

[Sou04] Souza, M. J. F. and Maculan, N. and Ochi, L. S. Metaheuristics. pages 659–672, Norwell, MA, USA, 2004. Kluwer Academic Publishers. http://www.decom.

ufop.br/prof/marcone/Publicacoes/SouzaMaculanOchi.pdf[Online; accessed 22-June-2011].

[Tho96] Thompson, J. and Dowsland, K. A. General Cooling Schedules for a Simulated Annealing Based Timetabling System. In Burke, E. K. and Ross, P., editor, *Practice and Theory of Automated Timetabling*, volume I of *Lecture Notes in Computer Science*, pages 345–363, Edinburgh, U.K., August/September 1996. 1st International Conference, PATAT 1995, Springer.

[Tur37] Turing, A. M. On Computable Numbers, with an Application to the Entscheidungsproblem. In *Proceedings of the London Mathematical Society*, volume 42 of *2*, pages 230–265, 1937. http://classes.soe.ucsc.edu/cmps210/Winter11/Papers/turing-1936.pdf[Online; accessed 10-July-2012].

[Wei09] Weise, T. Global Optimization Algorithms - Theory and Application -. http://www.it-weise.de/projects/book.pdf [Online; accessed 1-October-2011], June 2009.

[Whi01] White, G. M. and Xie, B. S. Examination Timetables and Tabu Search with Longer-Term Memory. In *Selected Papers from the Third International Conference on Practice and Theory of Automated Timetabling III*, PATAT '00, pages 85–103, London, UK, UK, 2001. Springer-Verlag. http://www.csi.uottawa.ca/~white/docs/lncs01.pdf [Online; accessed 22-June-2011].

[Wol02] Wolfram, S. *A New Kind of Science*. Wolfram Media, Inc., Champaign, Illinois, United States, 2002.