FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA

CENTRO DE INVESTIGACIONES EN INGENIERÍA
Y CIENCIAS APLICADAS

El Problema de la Programación de Cursos en una Universidad y una Propuesta de Solución

TESIS PROFESIONAL
PARA OBTENER EL GRADO DE:

DOCTORADO EN INGENIERÍA Y CIENCIAS APLICADAS
OPCIÓN TERMINAL TECNOLOGÍA ELÉCTRICA

PRESENTA:
Martín Heriberto Cruz Rosales

ASESOR: Dr. Marco Antonio Cruz Chávez

CUERNAVACA MOR. FEBRERO 2010

Resumen

Se presenta la investigación efectuada al problema de la programación de recursos y en específico se trata el problema de la Programación de Cursos en una Universidad. De esta investigación se hacen las siguientes tres aportaciones: La primera, una Formulación Matemática del modelo del problema el cual considera los recursos materiales y humanos que son limitados y costosos. También considera, las restricciones y necesidades para el uso o disponibilidad de esos recursos. Como segunda aportación, la creación e implementación de una Estrategia de Asignación de Recursos que tiene como criterio básico, realizar cada asignación de horario y espacio siempre y cuando se conserve la factibilidad de la solución. Para la aplicación de esta estrategia, primero se realiza una asignación de eventos al azar y posteriormente un reacomodo por intercambio de eventos. Comparada con otras estrategias aplicadas en la literatura, esta estrategía permite incrementar el porcentaje de soluciones factibles. Como tercera aportación, se diseñó e implementó un algoritmo que utiliza la metaheurística de Recocido Simulado para optimizar las soluciones obtenidas. Estas dos últimas aportaciones corresponden a la propuesta primaria de este trabajo para obtener una solución del problema de la Programación de Cursos en una Universidad.

A partir de los resultados obtenidos en las pruebas realizadas, se concluye que el algoritmo propuesto tiene muy buen desempeño y compite con los algoritmos representativos de la literatura ya que logra obtener un porcentaje mayor de soluciones factibles de las instancias problema consideradas grandes y también mejora la calidad de tales soluciones. La eficacia se demuestra en todas las pruebas realizadas con las instancias tomadas de los benchmarks existentes para el problema de Programación de Cursos en una Universidad. También es importante resaltar que el Algoritmo de Vecindad que se creó e implementó como perturbación en la metaheurística de Recocido Simulado constituye un buen aporte, dada la baja complejidad computacional y el buen desempeño demostrado durante la búsqueda de mejores soluciones.

Abstract

This thesis focuses on scheduling problems, specifically addressing the University Course Timetabling Problem. From the research conducted we can point to three major contributions: First, a mathematical formulation of the problem which considers limited and expensive issues such as human resources and material. The proposed formulation also considers restrictions and requirements for the use or availability of these resources. The second contribution is the creation and implementation of a Resource Allocation Strategy that has as a basic criterion, perform each allocation of timeslot and space as long as they keep the feasibility of the solution. For the implementation of this strategy a first random event assignment is performed allowing an events exchange thereafter. Compared with other strategies used in the literature, this strategy can increase the percentage of feasible solutions. A third contribution is the design and implementation of an algorithm that uses Simulated Annealing metaheuristic to optimize the solutions. These last two contributions correspond to the primary proposal of this work to obtain a solution of the University Course Timetabling Problem.

A series of tests were performed, from the results we can conclude that the proposed algorithm has a very good performance and can compete with algorithms reported in the literature. This is shown as our proposal is capable of producing a higher percentage of feasible solutions for the hardest instances taken from the existing benchmarks for the university course timetabling problem. Likewise, the quality of the solutions is considerably improved. It is important to note that the neighborhood structure created and implemented as a disturbance in the Simulated Annealing metaheuristic represents as well an important contribution, given its low computational complexity and its excelent performance in the search space to preserve the solution feasibility.

Nomenclatura

Nomenclatura General:

S	Espacio o conjunto de soluciones
s	Indica una solución, donde $s \in S$
$s*$	Indica un óptimo global o solución óptima
$N(s)$	N Es la vecindad de una solución s
$f(s)$	Función de costo de la solución s
$GSTT$	Siglas del algoritmo Generador de Soluciones de Timetabling (TT)
$\mathbb{R}$	Indicador de números reales

Nomenclatura del Modelo Matemático:

E	Conjunto de eventos. Un evento es sinónimo de clase, curso, examen, lección, conferencia, disertación, seminario. Entonces, se utiliza evento para generalizar una actividad académica donde participan estudiantes
n_E	Número total de eventos
T	Conjunto de timeslots. Un timeslot es un espacio de tiempo
D	Conjunto de días
P	Conunto de periodos, indica un espacio de tiempo
R	Conjunto de salones
n_R	Número total de salones
A	Conjunto de estudiantes (Alumnos)
n_A	Número total de estudiantes
F	Conjunto de facilidades que proporciona un salón
n_F	Número total de facilidades
FO	Siglas de Función Objetivo
$F1, F2, F3$	Funciones que contabilizan la ocurrencia de violaciones de las restricciones suaves $S1, S2, S3$ respectivamente.

Nomenclatura del GSTT, Metrópolis y Recocido Simulado:

i, j	Representan estados para el algoritmo de Metrópolis. Para el recocido simulado son soluciones
Ei	Energía del estado i
k_B	Indica la constante de Boltzman
L_k	Cadena de Markov de longitud k
T	Temperatura
I	Instancia de entrada, se refiere a una instancia problema que sirve como datos de entrada al GSTT u otro algoritmo
TT	Arreglo tridimensional donde se guardan los resultados de una solución
ExA	Arreglo tridimensional donde se conservan los eventos aun no asignados (eventos por asignar).
$ENA1, ENA2$	Arreglos tridimensionales para el movimiento de datos
$N(i)$	El entorno o vecindad de una solución i en el recocido simulado para el GSTT
To	Temperatura inicial para el recodido simulado en el GSTT
$\propto$	Función o coeficiente de reducción de la temperatura en el recocido simulado para el GSTT
ncM	Para una temperatura dada, indica el número de ciclos de Metrópolis o número máximo de iteraciones en recodido simulado para el GSTT
Tf	Temperatura final para el recocido simulado en el GSTT

Índice general

Índice de figuras

Índice de cuadros

Lista de Algoritmos

Capítulo 1

Introducción

La programación de recursos es una actividad que se necesita realizar en diferentes áreas de trabajo, como puede ser: en la distribución de recursos para la fabricación de objetos, en los centros laborales para la distribución de tareas al personal, en la calendarización de máquinas al establecer la secuencia de tareas a realizar, en la distribución de mercancía para colocarla en los puntos de venta, en la calendarización de los sistemas de transporte, en la secuencia de procesos en una computadora o cualquier otra máquina. Entonces, la programación de recursos es toda actividad que requiere de planeación, distribución o calendarización, donde son primordiales además de los recursos el tiempo, ya que es una variable importante a considerar para establecer la secuencia de actividades, procesos o tareas a realizar con esos recursos.

En el ambiente académico de cualquier nivel (primarias, secundarias, preparatorias, escuelas superiores, universidades, centros de investigación), en cada periodo escolar es necesario establecer horarios escolares. Estos horarios escolares pueden ser de diferentes tipos, por ejemplo: los horarios de clase teóricas, los horarios de laboratorio, los horarios de conferencias, los horarios de exámenes, los horarios de tutorías, etc. Realizar los horarios implica considerar el tiempo en que se efectuarán las clases, las conferencias los exámenes, las tutorías. Estas actividades (clases, conferencias, exámenes, tutorías) son recursos abastractos, que para establecer el horario y la secuencia en que se efectuarán, se requerirá considerar otros recursos tangibles, como pueden ser: los salones donde se efectuarán las clases, los auditorios en donde se llevarán a cabo las conferencias, los

espacios donde se efectuarán los exámenes, el lugar donde se atenderán las tutorías. También habrá que considerar los recursos humanos como: los alumnos que asistirán a cada clase, las personas que asistirán a las conferencias, los estudiantes que efectuarán los exámenes, los alumnos que asistirán a las tutorías, los profesores que atenderán las clases, investigadores que participarán en ponencias como expositores o como asistentes, el personal de apoyo para las diferentes actividades. Además, cada una de estas actividades requieren de diferentes tipos y cantidad de otros recursos, como pueden ser: proyectores, pintarrones, atriles, marcadores, computadoras, periféricos, conexiones vía internet, sistemas de comunicación diversos, mesas, sillas, escritorios, cortinas, iluminación, etc. Por lo tanto, para desarrollar estas actividades escolares hay que considerar los recursos humanos y materiales necesarios y establecer los horarios y la secuencia en que se efectuarán tales actividades.

Crear o diseñar los horarios, no es siempre una tarea sencilla, y dependiendo del nivel escolar para el que se pretenda establecer el diseño de los horarios, de esa magnitud de complejidad será el nivel del diseño, implementación y complejidad computacional. Para una escuela primaria es sencillo el diseño de los horarios, ya que normalmente se cuenta con una planta fija de profesores, una cantidad y tamaño de salones adecuados para las clases y una cantidad más o menos establecida de estudiantes que atenderán las clases, los recursos complementarios que se utilizan a nivel primaria están también ya establecidos. En resumen, para el nivel escolar de primaria, los recursos necesarios en cada periodo escolar son casi siempre los mismos. En nivel secundaria, sigue siendo sencillo establecer los horarios escolares; pero ya se presenta una complejidad mayor, ya que por ejemplo, la distribución de profesores para dar clases requiere considerar el dominio académico de cada profesor, que el tipo de clases puede ser teórica, práctica o de laboratorio, la disponibilidad de tiempo de cada profesor y que las clases están restringidas a un periodo de tiempo normalmente de una hora de exposición. Sin embargo, conforme el nivel escolar aumenta se dificulta la programación de los horarios. A nivel superior o universitario y sobre todo a nivel posgrado, obtener los horarios escolares se convierte en una tarea tan difícil que el diseño se vuelve un problema complicado debido a las restricciones y necesidades individuales y de conjunto que hay que considerar para

la programación de los recursos materiales y humanos.

Para tratar la problemática de obtener los horarios escolares del nivel universitario y posgrado, en la literatura se propuso un modelo textual del problema de la programación de cursos y también instancias problema o benchmarks que se utilizan como paradigmas para tratar de obtener de ellos soluciones que sean factibles. El nombre del problema modelo es Programación de Cursos en una Universidad (o UCTP por sus siglas en inglés de University Course Timetabling Problem). El problema de la Programación de Cursos en una Universidad tiene como finalidad básica incluir en algún espacio recursos y/o distribuir en el tiempo los horarios de actividades, generalmente llamados eventos (cursos, clases, exámenes, lecciones, tareas), en una tabla de horarios formada de un número limitado de espacios (tipo casillas o celdas) llamados timeslots (periodos de tiempo u horarios). A esta actividad de distribución y alojamiento de eventos en los timeslots se le llama asignación y a todo el proceso de asignación se le da el nombre de Programación de Cursos en una Universidad, ya que se hace la distribución considerando las restricciones que implica el tratamiento de los recursos humanos y materiales que son limitados y a las condiciones para establecer los horarios y la secuencia de cursos. En la literatura se puede encontrar información sobre modelos del UCTP, tablas de horarios y algoritmos de solución de los benchmarks. Por ejemplo, en [Schaerf, 1999] se puede encontrar una revisión muy completa de varios tipos de calendarizadores (timetabling) y sus formulaciones con algoritmos genéticos, tabu search, y satisfacción de restricciones; en [Zervoudakis and Stamatopoulos, 2001] se puede ver la construcción de un calendarizador de cursos para instituciones académicas con un modelo en términos de un programador de restricciones; en [Rossi-Doria et al., 2003] se encuentra un estudio muy completo con modelos de: Algoritmos Genéticos, Colonia de Hormigas, Búsqueda Local Iterada, Recocido Simulado y Búsqueda Tabú; en [Burke et al., 2004] se presenta un modelo en base a Búsqueda Tabú y con aprendizaje reforzado; en [Kostuch, 2005] se presenta un modelo para el UCTP con un enfoque que le llaman Tres Fases, la primera fase es la construcción de una solución factible, en la segunda fase utilizan Recocido Simulado para ordenar los timeslots creados y en la tercera fase utilizan también el Recocido Simulado con un intercambiador (estructura de vecindad) de eventos para mejorar

la solución.

En este trabajo doctoral se hizo una investigación sobre la problemática de determinar los horarios de cursos y en específico se trató el problema de la Programación de Cursos en una Universidad. De la investigación realizada se hicieron las siguientes tres aportaciones: una Formulación Matemática del modelo del problema el cual considera los recursos materiales y humanos a programar o calendarizar, la creación e implementación de una Estrategia de Asignación de Recursos que sirve para hacer una mejor asignación de horarios y/o una mejor distribución de los espacios en la creación de las tablas de horarios a partir de los datos de entrada formados por las instancias problema (benchmarks) y el diseño e implementación de un algoritmo que se le llamó GSTT. La Estrategia de Asignación de Recursos y el algoritmo GSTT corresponden a la propuesta de solución del UCTP que se planteó originalmente en este trabajo doctoral. También se creó e implementó un algoritmo de búsqueda local que se incluyó como perturbación en la metaheurística de Recocido Simulado para la búsqueda de mejores soluciones en la etapa de optimización. Como se verá en el análisis de resultados (Capítulo 6), las soluciones factibles que se obtienen con el algoritmo GSTT son muy buenas en eficacia y eficiencia y superan los resultados reportados por los algoritmos de la teoría. Además, el resultado de las pruebas también indica que el algoritmo GSTT es más efectivo que las cinco metaheurísticas de la teoría en cuanto al porcentaje de soluciones factibles que puede obtener. Aunque dentro de nuestro objetivo estaba tratar solo con instancias grandes, también se hicieron pruebas con instancias medianas. Los resultados con estas instancias resultaron satisfactorios y competitivos con los resultados que se reportan en la teoría.

A continuación se da la entrada de la investigación que se realizó en este trabajo doctoral sobre el problema de la Programación de Cursos en una Universidad, se trata la problemática de la calendarización, el objetivo de la investigación, su alcance y las contribuciones que se obtienen del diseño e implementación de la propuesta de solución. Para finalizar el capítulo, se da la organización en que se presenta este trabajo.

1.1. Problemas de Optimización Combinatoria

Los problemas de Optimización son aquellos que tratan con un conjunto de instancias. Donde el término instancia y problema tienen una connotación diferente. En una instancia se da una entrada de datos para obtener una solución y un problema es una colección de instancias donde usualmente todas estas instancias están generadas con la misma estructura. También, los problemas de Optimización tratan con variables de tipo continuo o combinatorio. Las variables continuas son las que se manejan con números reales y las variables combinatorias son variables discretas ya que manejan números enteros. Entonces, debido al tipo de variables enteras que maneja la Optimización Combinatoria también se le conoce como Optimización Discreta. Por lo tanto, se tendrán dos tipos de Problemas de Optimización que se clasificarán de acuerdo al tipo de variables como: Problemas de Optimización Continua y Problemas de Optimización Discreta (o combinatoria). Considerando las variables enteras con que se trabaja en la optimización, algunos autores prefieren llamarla Optimización Combinatoria y otros Optimización Discreta; de cualquiera de las dos formas se hará referencia a lo mismo.

En el área de la optimización combinatoria o discreta resolver un problema consiste en encontrar la mejor solución entre un número finito o infinito numerable de soluciones alternativas [Papadimitriou and Steiglitz, 1982]. Los problemas en la optimización combinatoria se pueden y deberían ser formulados de manera no ambigua utilizando notación y terminología matemática [Papadimitriou and Steiglitz, 1998]. Además, se presupone que el conjunto de soluciones sea finito, que las soluciones son cuantificables y que la calidad de la solución depende del valor que se obtenga, el cual puede ser comparado con cualquier otra solución [Aarts and Korst, 1989]. Entonces, en la optimización combinatoria existen diferentes soluciones y un criterio para discriminar entre ellas. El problema consiste en encontrar el valor de las variables de decisión de una función objetivo, la cual alcanza su valor máximo o mínimo dependiendo de las variables antes mencionadas y que también están sujetas a ciertas restricciones.

Existe una gran cantidad de problemas de optimización en la industria, en las ciencias y en otros ámbitos como el comercio, servicios y administración. Entre los problemas de

optimización combinatoria más comunes y que son muy referenciados están: el problema del agente viajero (TSP por sus siglas en inglés de Travelling Salesman Problem), el problema general de asignación, y los clásicos de calendarización muy conocidos por sus dos nombres en inglés de “Scheduling” y “Timetabling”. Hasta ahora no se conoce un algoritmo determinístico capaz de generar la solución óptima de estos problemas en tiempo polinomial, por lo que se considera que están entre los problemas computacionales más difíciles de resolver. En el ámbito computacional y de investigación, a estos problemas se les califica por su complejidad computacional con el nombre de NP-Duros, particularmente en el contexto de la teoría de la NP-Completez [Martí, 2003].

Cuando se habla de complejidad computacional y de los problemas NP, se tiene que hablar del problema de satisfacibilidad (que se satisface). El problema de satisfacibilidad (o simplemente SAT) se define de la siguiente manera:

> Definición: *Dadas* m *cláusulas* $C_1, C_2, \ldots, C_m$ *las cuales emplean las variables booleanas* $x_1, x_2, \ldots, x_n$ *¿es satisfacible la fórmula* $C_1 \wedge C_2 \wedge \ldots \wedge C_m$*?*

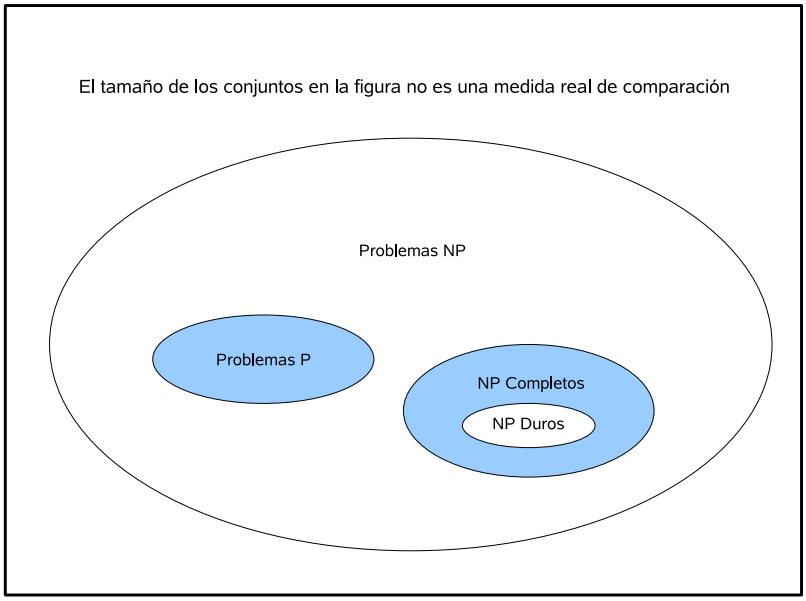
En SAT cada restricción se expresa mediante una ecuación booleana (aquella compuesta por variables que pueden asumir únicamente los valores de V o F) llamada cláusula proposicional, y el conjunto de cláusulas establece una fórmula booleana. De esta forma, el problema consiste en determinar una asignación de valores de verdad para las variables que satisfagan la fórmula proposicional (que la hagan verdadera), es decir, que satisfagan todas las restricciones.

El problema de satisfacibilidad es sencillo de enunciarse y su solución puede verse de manera combinatoria al tratar todas las posibles asignaciones de valores de verdad a los enunciados de un problema y verificar si se satisface la formulación. Sin embargo, este no es un algoritmo eficiente, ya que hay 2^m asignaciones de verdad que probar, con una elección binaria para cada variable. Por lo que el tiempo de ejecución del algoritmo crece exponencialmente con el tamaño de la entrada del problema.

Stephen Cook presentó la teoría de la NP-Completez y también demostró que SAT era un problema NP completo, entonces SAT pasó a ser el primer problema en ser demostrado como NP completo [Cook, 1971]. Con este estudio, Cook constituyó la piedra angular de la teoría de la complejidad computacional [Papadimitriou and Steiglitz, 1982, Papadimitriou, 1994]. Cook probó que cualquier problema de la clase NP puede ser reducido al problema SAT a través de una transformación de tipo polinomial. Esto significa que si el problema SAT se pudiera resolver en un tiempo polinomial de cálculo, todos los problemas no polinomiales también podrían resolverse en tiempo polinomial, lo que significaría que la clase NP dejaría de existir. De esta forma, si cualquier problema en NP es intratable, entonces SAT es un problema intratable. Cook también sugirió que el problema de satisfacibilidad y otros problemas NP tenían la característica de ser los problemas más difíciles. En términos sencillos, se puede concluir que un problema duro o difícil es aquel para el que no podemos garantizar el encontrar la mejor solución posible en un tiempo de computación razonable. Actualmente se acepta que P es un subconjunto de NP [Papadimitriou and Steiglitz, 1982] y se ha demostrado de forma trivial [Cook, 2000]. La pregunta que aun queda en el aire sin respuesta es si $P = NP$. En el Cuadro 1.1 se muestra un diagrama de la clasificación de los problemas según la teoría de la complejidad.

Muchos de los problemas dentro del área de optimización combinatoria son problemas de secuenciación y programación -o scheduling y timetabling como se conocen en Inglés-, problemas de toma de decisiones que tienen como meta la optimización de uno o más objetivos. El objetivo primordial de los problemas de secuenciación y programación es distribuir y asignar las actividades de tal forma que se usen los recursos disponibles de forma eficiente [Pinedo, 2002].

Algunas variedades de los problemas de programación (scheduling o timetabling) de recursos, que básicamente tienen una estructura de planteamiento similar al UCTP, son: la Programación de Empleados (ETP por sus siglas en inglés de Employee Timetabling Problem), el problema Programación Educacional (ETT por sus siglas en inglés de Educational Timetabling), la Asignación de Actividades de Enfermeras (NRP por sus siglas



Cuadro 1.1: Clasificación de problemas según la teoría de la complejidad

en inglés de Nurse Rostering Problem) y el problema de Itinerario de Trenes (Railway Timetabling).

Encontrar la solución de problemas NP resulta difícil si se quiere obtener una solución exacta y el tiempo de computación resulta muy alto si no imposible de calcular en tiempo polinomial. Sin embargo, existe una gran cantidad y variedad de problemas difíciles, que aparecen en la práctica y que requieren ser resueltos de forma eficiente. Este requerimiento ha hecho necesario desarrollar métodos y procedimientos para encontrar buenas soluciones aunque no necesariamente sean exactas u óptimas. Estos métodos, en los que la rapidez del proceso es tan importante como la calidad de las soluciones obtenidas, se denominan métodos *heurísticos o aproximados* [Colorni et al., 1998].

Para la programación de recursos y en particular para la Programación de Cursos en una Universidad, los métodos heurísticos ofrecen una buena alternativa para encontrar soluciones en tiempos polinomiales. Al hablar de heurísticas, existen algunos criterios que se deben considerar en el diseño. Una idea fundamental dentro de los métodos heurísticos es que lo más *corto/barato* es lo más *rápido/mejor*. Entonces, si existe un criterio de

aceptación (tolerancia) se habla de un problema de semi optimización, ya que se busca acortar el proceso para no exceder los tiempos de computación innecesariamente. Cuando se habla de alta probabilidad, se tiene un problema de optimización aproximada, ya que se estará considerando alargar el ciclo de proceso para obtener mejor resultado. En la práctica, se busca establecer un balance razonable entre calidad de la solución (mejor solución) y costo de la solución. Algo un poco más ambicioso en el diseño e implementación de los algoritmos de solución es dotarlos con parámetros ajustables para cambiar el compromiso entre calidad y costo.

Es necesario mencionar que las heurísticas cuando se diseñan con una estructura formal, en su aplicación se podrán ejecutar de forma iterativa e intensiva; pero si además, su utilización se hace extensiva por su reconocida utilidad en la búsqueda y mejora de soluciones, entonces se consideran como metaheurísticas. Las metaheurísticas más utilizadas para tratar el UCTP han sido: Optimización con Colonia de Hormigas, Búsqueda Local Iterada, Algoritmos Genéticos, Recocido Simulado y Búsqueda Tabú. En el capítulo 4 se verán estas metaheurísticas con sus algoritmos básicos para tratar el UCTP.

1.2. La Programación de Cursos

El problema de Programación de Cursos en una Universidad, pertenece al ámbito de los problemas de Programación del área de Optimización Combinatoria y del problema General de Asignación. Es decir, haciendo una categorización, el UCTP es un subproblema del problema de Programación (o Calendarización), y a su vez el problema de Programación es un subproblema del problema de Asignación. Además, el UCTP es un problema de clase NP completo que se puede resolver con métodos exactos o heurísticos. Sin embargo, solo es factible obtener una solución exacta para instancias pequeñas; pero si el tamaño de la instancia aumenta, el tiempo de ejecución de un algoritmo exacto aumenta de forma exponencial. Para aplicaciones prácticas se emplean métodos aproximados que obtienen soluciones próximas a la solución óptima con tiempos de cómputo razonables, aunque no se pueda garantizar alcanzar la solución óptima

En el problema de la Programación de Cursos en una Universidad existen diversos tipos de restricciones; pero el modelo más utilizado para estudio es el que tiene dos tipos de restricciones, y que fue introducido por Ben Paechter [Rossi-Doria et al., 2003] para mostrar aspectos reales del problema de calendarización. Este modelo fué tomando como patrón por la Universidad de Napier [Schaerf, 1999, Chiarandini and Stützle, 2002]. Los dos tipos de restricciones son:

- Restricciones duras. Son restricciones que se deben cumplir obligatoriamente para que se obtenga una solución factible. Las variables de estas restricciones están relacionadas con los recursos físicos. Por ejemplo, en un salón y en un periodo de tiempo definido, sólo se puede efectuar un evento a la vez.

- Restricciones suaves. Son restricciones que es preferible cumplirlas; pero que no es necesario que se cumplan para que la solución sea factible. Se consideran restricciones de preferencia. Las faltas a estas restricciones se cuantifican para dar la calidad de la solución. Ejemplo, Un alumno no debería tener solo una clase al día.

Los primeros intentos en automatizar la programación de horarios se hicieron por 1960, donde se utilizaron técnicas basadas en simular el diseño por medio de la simulación del proceso manual [Appleby et al., 1960] para obtener los horarios. La idea básica de esta técnica consistía en programar primero las clases con mayor grado de restricciones.

En la década de los 70's se empezaron a aplicar técnicas más genéricas para resolver el problema. Algunos de los estudios más conocidos son: con algoritmos basados en programación entera [Smith, 1975], flujo de redes [de Werra, 1985, de Werra et al., 1985, Cheng et al., 2003] por cierto, de Werra fue de los primeros en presentar una formulación matemática del problema. También se ha abordado el problema como una transformación al problema del coloreo de grafos [Brelaz, 1979, Burke et al., 1994], donde los vértices representan los eventos y las aristas los conflictos existentes. Posteriormente se presentan enfoques basados en técnicas de búsqueda con heurísticas, algunos de ellos son: con satisfacción de restricciones [Faber et al., 1998], con recocido simulado [Elmohamed and Fox, 1997, Abramson, 2001], con algoritmos genéticos hay varios, una

buena referencia es [Abramson and Abela, 1992] con búsqueda tabú [Hertz, 1992]. Una referencia que incluye cinco metaheurísticas donde se presenta una buena explicación y reporte de pruebas se puede ver en [Rossi-Doria et al., 2003]. Los algoritmos de estas heurísticas de optimización son genéricos e independientes del problema que tratan de optimizar y se les conoce bajo el nombre de metaheurísticas [Blum and Roli, 2003]. La ventaja de las metaheurísticas radica en que una pequeña modificación o adaptación con respecto de la formulación general es suficiente para que puedan ser aplicados a un problema concreto.

1.3. Objetivo de la Investigación

El objetivo de este trabajo doctoral es: El diseño de una metaheurística eficiente y eficaz para instancias grandes del problema de la Programación de Cursos en una Universidad.

Para llevar a efecto este objetivo, se investigará sobre el problema de la Programación de Cursos en una Universidad y se diseñara un algoritmo de solución que obtenga soluciones factibles del problema, las cuales deberán ser competitivas con los resultados reportados por los algoritmos de la teoría.

La motivación para realizar esta investigación es que, en los planteles escolares de nivel superior cada inicio del periodo escolar se presenta la necesidad de planificar, organizar y distribuir los horarios de clases. Esta tarea tiene que considerar las diferentes restricciones y necesidades de los recursos humanos y materiales disponibles. Por ejemplo, las necesidades de los alumnos según sus niveles escolares y los diversos planes de estudio existentes, las preferencias de los profesores según su dominio académico y sus posibilidades de tiempo, la disponibilidad de los espacios y tamaño para impartir las clases.

Hasta la fecha, la planificación y distribución de los recursos para elaborar los horarios escolares de los planteles se ha realizado a mano, solo algunos planteles cuentan con

algún tipo de herramientas de software para obtener los itinerarios o tablas de horarios; pero no son sistemas verdaderamente automatizados y completos. Son muy contados los planteles que han abordado el problema de obtener los horarios utilizando sistemas automatizados inteligentes, uno de ellos es por ejemplo: del sistema de tecnológicos ITESM en los campus de Nuevo León y Morelos, otras dos universidades que han tratado son la Universidad Autónoma de Yucatán y la Universidad Juárez Autónoma de Tabasco; pero lo han hecho de forma reducida (local) considerando pocos recursos (instancias pequeñas). Esto quizá es debido a la complejidad inherente (problema NP) que existe para obtener los horarios escolares a nivel universitario y posgrado. Además, hay que considerar que una solución que sea factible y se aplique en alguna etapa escolar, no se puede aplicar a las siguientes fases, ya que las restricciones y necesidades cambian en cada periodo escolar. En el mercado existen algunos calendarizadores que resuelven parcialmente el problema de obtener las tablas de horarios; pero desgraciadamente cada plantel tiene diferentes necesidades y restricciones y en la mayoría de casos estos sistemas diseñados de forma genérica no cumplen con las necesidades particulares de cada institución.

1.4. Alcance de la Investigación

De la investigación sobre el tema doctoral propuesto, a continuación se mencionan las actividades realizadas.

- Se hizo una revisión extensiva de la literatura existente sobre el problema de la Programación de Cursos en una Universidad y la literatura relacionada con el tema. En este trabajo se mencionan las más representativas
- Se estudiaron los algoritmos existentes en la literatura y los que se han utilizado para tratar de resolver el UCTP.
- Se hizo una búsqueda también en la literatura y en los sitios universitarios sobre los benchmarks existentes que sirvieran de paradigma para probar y ejecutar los algoritmos que se pudieran obtener de la literatura.
- Se consiguieron los programas ejecutables de los cinco algoritmos representativos en la literatura para obtener soluciones del UCTP, estos programas se ejecutaron

para reproducir los resultados que reportan en la literatura y hacer comparaciones con el algoritmo que se desarrolló en el laboratorio del CIICAp-UAEM.

- Se obtuvo una formulación Matemática del modelo problema de la Programación de Cursos en una Universidad.

- Se desarrolló e implementó un algoritmo como propuesta de solución del UCTP. El diseño se enfocó para obtener soluciones de instancias problema (benchmarks) grandes.

- Se desarrolló e implementó un algoritmo de búsqueda local que se integró en la metaheurística de Recocido Simulado, la cual se escogió para tratar de mejorar las soluciones que se obtienen de los benchmarks. El algoritmo se diseño para realizar movimientos de intercambio sencillos.

1.5. Contribución

Con esta investigación se desea contribuir en la obtención de los horarios escolares a nivel superior y de posgrado. El lograr obtener soluciones satisfactorias del UCTP implica que la mayoría de los estudiantes, si no todos, queden satisfechos con la distribución de los horarios que cumplen sus necesidades escolares, también se satisfacerá la escuela al poder obtener una distribución adecuada de los recursos materiales, y así cumplir con las necesidades de clases, espacios, horarios y disponibilidad de los profesores. El obtener horarios que cumplan las restricciones y las necesidades de todo el personal involucrado tiene además una gran ventaja humanista y social, ya que la gran mayoría, si no todos, quedan contentos y esto ayuda también en el rendimiento universitario.

De la investigación realizada, se hicieron las siguientes aportaciones:

- Como primera aportación, la Formulación Matemática del modelo del problema, que considera los recursos humanos y materiales así como las restricciones de tiempo y espacio para programar los recursos.

- Una segunda aportación es la creación de una Estrategia de Asignación de Recursos que tiene como criterio básico para la distribución de horarios y espacios, el de asignar los recursos siempre y cuando se conserve la factibilidad de la solución.
- Como tercera aportación, se diseño e implementó un algoritmo generador de soluciones factibles del UCTP que se le llamó GSTT, el cual utiliza la metaheurística de Recocido Simulado para optimizar las soluciones obtenidas. Estas soluciones o producto representan las tablas de horarios de cursos (course timetabling)
- Como cuarta aportación, se desarrolló e implementó un algoritmo para la búsqueda local de soluciones en una vecindad. Este algoritmo, se integró como perturbación en la metaheurística de Recocido Simulado.

Para los casos reales, los algoritmos aquí presentados se podrían configurar para adecuarlos según las necesidades particulares de cada centro escolar.

1.6. Organización de la Tesis

Esta tesis se desarrollo en el Centro de Investigación en Ingeniería y Ciencias Aplicadas (CIICAp) de la Universidad Autónoma del Estado de Morelos (UAEM) bajo la dirección del Dr. Marco Antonio Cruz Chávez, investigador del CIICAp y miembro del cuerpo académico en consolidación “Optimización y Software”.

En este capítulo uno de Introducción, se hace la presentación de la problemática para obtener los horarios escolares a nivel universitario y la semblanza de como se pretende abordar el problema. Se presenta el objetivo, el alcance y la contribución del trabajo doctoral.

El capítulo dos trata el Problema de la Programación de Cursos. Se presentan los métodos conocidos para la Búsqueda Local de Soluciones y como se obtiene la Función de Costo. Posteriormente se da el planteamiento del UCTP y su representación simbólica. Para terminar, se explica el funcionamiento de una estructura de vecindad.

En el capítulo tres se presenta la formulación Matemática del modelo del problema. También, se da el detalle de las expresiones matemáticas.

En el capítulo cuatro se presentan los métodos más conocidos para resolver el UCTP.

El capítulo cinco muestra la Propuesta de Solución para el UCTP. Se da la representación del modelo, la aplicación de la estrategia de asignación de recursos y el diseño del algoritmo GSTT, se hace una explicación detallada de los módulos que componen el algoritmo. Posteriormente, se incluye el algoritmo de recocido simulado que mejora las soluciones que se obtienen.

El capítulo seis de Análisis de Resultados. Se dan los parámetros de sintonía obtenidos del análisis de sensibilidad, las pruebas experimentales efectuadas y el análisis de efectividad de los resultados obtenidos. También, la comparación de los resultados obtenidos con el GSTT contra las metaheurísticas utilizadas en el estado del arte.

Para terminar, en el capítulo siete se dan las Conclusiones del trabajo de investigación y el Trabajo Futuro que se desea llevar a cabo dentro de la línea de investigación aquí tratada.

Capítulo 2

El Problema de la Programación de Cursos

En la literatura se puede encontrar el nombre del problema de la Programación de Cursos como: Programación de horarios, Asignación de Cursos o Calendarización de Cursos. De las cuatro formas se refieren al mismo problema.

La programación o calendarización de cursos describe un problema que usualmente surge en el ambiente educacional (escuelas, universidades, conferencias, seminarios) cuando se requiere planificar y organizar los recursos escolares (clases, seminarios, exámenes, pláticas, salones, estudiantes, profesores, Etc.) en los espacios y en el tiempo. El desarrollo de la programación, es la tarea de colocar entidades generalmente llamadas eventos (cursos, clases, exámenes, lecciones, tareas) en una tabla de horarios formada de un número limitado de unidades llamadas timeslots (periodos de tiempo). En la literatura se puede encuentrar información sobre la problemática de la programación de cursos, de los algoritmos de solución y de las tablas de horarios. En [Schaerf, 1999] se encuentran variaciones del problema de programación de cursos y formulaciones con técnicas como algoritmos genéticos, tabu search, y satisfacción de restricciones; en [Zervoudakis and Stamatopoulos, 2001] se puede ver la construcción de un calendarizador de cursos para instituciones académicas con un modelo en términos de un programador de restricciones; en [Rossi-Doria et al., 2003] se encuentra un estudio muy completo con modelos de: Algoritmos Genéticos, Colonia de Hormigas, Búsqueda Local

Iterada, Recocido Simulado y Búsqueda Tabú; en [Burke et al., 2004] se presenta un modelo en base a Búsqueda Tabú y con aprendizaje reforzado.

Un caso particular del Problema General de Asignación de Recursos es el Problema de la Programación de Cursos en una Universidad (UCTP por sus siglas en Inglés: University Course Timetabling Problem). Las características de los componentes que se consideran en este problema, como recursos humanos y materiales, facilidades y necesidades, así como las restricciones, hacen difícil de obtener soluciones satisfactorias en tiempos que sean adecuados y aceptables para implantar en un centro escolar.

Para ser más explícitos, el UCTP consiste en asignar un conjunto de eventos (cursos, clases, seminarios, exámenes, pláticas, lecciones, tutoriales, etc.) y recursos (salones, laboratorios, estudiantes, profesores, máquinas, computadoras, Etc.) dentro de un número limitado de periodos de tiempo (timeslots, horarios, sesiones), tal que se satisfaga un objetivo y un conjunto de restricciones. Además se necesita que el objetivo se cumpla en un espacio de tiempo de computación razonable al tratar de obtener una solución que sea satisfactoria. Por ejemplo, obtener una solución no debe tomar tiempos de cálculo exponenciales o que la solución sea de mala calidad (que afecte a demasiados individuos). También, se debe considerar que la cantidad y tipo de recursos así como las posibilidades y necesidades de los recursos humanos son diferentes en cada periodo escolar. Por lo tanto, no se puede poner en práctica la misma solución para cada periodo escolar.

En las secciones siguientes se expone y se da una explicación más amplia del problema de la programación de cursos y del ámbito donde se aplica. Se indica cual es el espacio de búsqueda para poder encontrar soluciones y los métodos de búsqueda que se utilizan en la literatura, así como la evaluación de la función de costo. También, se incluye el plantamiento del UCTP con la mención de los recursos y descripción de las restricciones del problema. Además, la representación simbólica con las estructuras de datos que se utilizan para manejar el proceso, junto con la estructura de vecindad y búsqueda local más utilizada para la aplicación del UCTP.

2.1. La Optimización Combinatoria

Cualquier problema de programación (ídem calendarización) de recursos pertenece al área del Problema de Optimización Combinatoria. En general, un problema de optimización combinatoria tiene un espacio de búsqueda discreto finito S, y una función f, que mide la calidad de cada solución en S.

$$f : S \rightarrow \mathbb{R} \tag{2.1}$$

El problema es encontrar

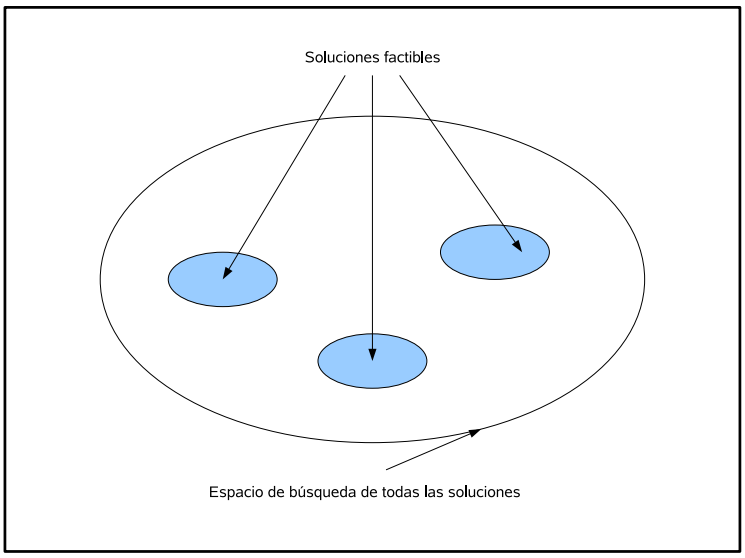
$$s* = arg\min_{s \in S} f(s) \tag{2.2}$$

Donde s es un vector de variables de decisión y f es la función de costo. El vector $s*$ es un óptimo global. La vecindad $N(s)$ de una solución s en S está definida como el conjunto de soluciones que pueden obtenerse de s por un *movimiento*. Cada solución se le llama un vecino de s.

Para cada s no necesita ser listado explícitamente el conjunto $N(s)$, en general se define implícitamente por referencia a un conjunto de movimientos posibles. Los movimientos están definidos usualmente como modificaciones locales de alguna parte de s. Entonces, los movimientos en una vecindad es uno de los ingredientes clave en la búsqueda local de soluciones. Sin embargo, en la definición anterior no hay implicación de que exista ”cerradura” en algún sentido entre vecinos, y actualmente también se pueden usar definiciones de vecindad. Este operador puede ser absolutamente complicado y puede ser incluso una metaheurística.

2.2. Espacio de Búsqueda

Cuando se trabaja con dominios discretos es posible definir el espacio de búsqueda en términos de los posibles valores que cada variable puede tener [Melício et al., 2003]. Para este problema se define el conjunto $T = \{t_1, \ldots, t_p\}$, como el conjunto de posibles valores (timeslots) que cada curso o evento puede tener (donde se puede asignar).



Cuadro 2.1: Espacio de búsqueda de todas las soluciones

Definición de Espacio de Búsqueda: *El conjunto completo de soluciones que pertenece al espacio de búsqueda se define por* $S = T_1 * \ldots * T_l$*. Si todas las* T_i *son iguales, entonces* $S = T^l$ *y* $|S| = |T|^l = p^l$*.*

Este valor es un caso extremo. Por ejemplo, hay 10^{10} posibles soluciones si hay 10 eventos y 10 timeslots. Aun si se restringe cada evento a un timeslot diferente se tendrá 10! = 3;628,800 posibles asignaciones. Como puede verificarse fácilmente, el espacio de búsqueda para este tipo de problema es muy grande. Sin embargo, no todas las soluciones son factibles, por ejemplo, una solución factible tiene que tener todos sus eventos programados y satisfacer un cierto número de restricciones (restricciones duras). Un espacio de búsqueda posible para este tipo de problema puede ser similar al que se muestra en el Cuadro 2.1.

Para ciertos problemas es muy difícil conocer de antemano si existe al menos una solución factible antes de iniciar cualquier algoritmo de búsqueda. Así, cualquier algoritmo de búsqueda debe poder transcurrir a través del espacio de búsqueda, incluso entre regiones infactibles. Una de las formas más comunes de hacer esto es *penalizar las restricciones que no se satisfagan y contabilizarlas en una función de costo*.

2.3. Métodos de Búsqueda Local

Cualquier algoritmo de búsqueda local inicia con una solución inicial y luego continuamente trata de encontrar una mejor solución buscando en la vecindad de la solución actual. Un proceso local se puede ver como un camino en un grafo $G = (S, C)$ donde el conjunto de vértices es el conjunto de soluciones S y existe una cuerda (s, s') en C sí y solo sí s y s' son vecinos es decir: $s, s' \in N(S)$. La eficiencia de cualquier método de búsqueda local depende del modelado [Hertz and Widmer, 2003]. La sintonía de los parámetros nunca balanceará una mala definición del conjunto de soluciones, de la vecindad o de la función de costo. En general las siguientes reglas son aplicables para cualquier método de búsqueda local.

1. Debe ser fácil generar soluciones en S.
2. Para cada solución, debe haber una trayectoria que conecte hacia una solución óptima.
3. La solución en la vecindad debe estar en un cierto sentido cerca de s (fuertemente correlacionada a s).

En toda búsqueda local es importante controlar el desarrollo del proceso para evitar quedar atrapado en mínimos locales. En el estado del arte existen varias alternativas que se han propuesto para controlar el proceso, entre las formas más aplicadas [Rossi-Doria et al., 2002, Rossi-Doria et al., 2003, Melício et al., 2003] se pueden encontrar las siguientes:

- Ejecutar el algoritmo de búsqueda local a partir de muchos puntos iniciales.
- Introducir restricciones adicionales a la función objetivo.
- Introducir una estructura de vecinos más compleja para buscar en regiones más grandes del espacio.
- Alterar soluciones buscando diversidad.
- Introducir mecanismos de memoria para evitar regresar a estados visitados.

1.Seleccionar aleatoriamente un evento e_i

2.Seleccionar aleatoriamente un nuevo timeslot del conjunto *T* (de timeslots)

3.Verificar las restricciones duras:

3.1 Si satisface las restricciones duras, asignar el evento e_i en el timeslot seleccionado en 2

3.2 Si no satisface las restricciones duras, deje el evento e_i fuera de la tabla de horarios (del timeslot seleccionado en 2)

Cuadro 2.2: Algoritmo de movimiento individual

Por lo tanto, es importante definir vecindades donde sea posible encontrar la mejor solución dentro de una cantidad de tiempo que sea razonablemente pequeña. Para cumplir con este objetivo y considerando algunas de las alternativas antes propuestas, a continuación se presentan varios movimientos sencillos que se han utilizado con éxito en los métodos de búsqueda local.

2.3.1. Movimiento individual

Debido a su simplicidad este tipo de movimiento desde su creación lo han adoptado muchos autores. El algoritmo consiste de instrucciones sencillas las cuales se muestran en el Cuadro 2.2.

Este tipo de movimiento es idéntico a varios de los movimientos descritos en la literatura, por ejemplo en [Elmohamed and Fox, 1997, Reeves, 1999, Thompson and Dowsland, 1996]. A pesar de la sencillez de los movimientos del algoritmo, como se menciona en [Abramson, 2001], este método de búsqueda permite una distribución bien balanceada de eventos entre todos los timeslots. El tamaño de esta vecindad es,

$$|N(s)| = E * (T - 1) \tag{2.3}$$

Donde E es el número de eventos y T es el número de timeslots.

Es importante hacer notar que en la forma clásica de hacer la distribución de eventos en el espacio de asignación, en cada timeslot (horario o periodo de tiempo) quedan programados uno o más eventos -entre más grande la instancia más eventos en un timeslot-. La distribución de los eventos en los timeslots, depende también del tipo de recurso, por ejemplo: el salón se requiere para clase teórica o laboratorio y cuáles son sus facilidades.

2.3.2. Movimiento individual con mejora heurística

Algunos autores sugieren que probablemente sería mejor transitar a través del espacio de búsqueda solo entre óptimos locales [Martin and Otto, 1996, Lourenço et al., 2001, Lourenço et al., 2002]. Sin embargo, puesto que se aplica muchas veces el movimiento, esta mejora heurística debe ser fácil de calcular. Con esto en mente se define el algoritmo basado en el movimiento individual con una mejora heurística sencilla, ver el algoritmo en el Cuadro 2.3.

2.3.3. Doble movimiento

Este tipo de movimiento, también llamado intercambio de pares (pairwise), se usa en muchos tipos de problemas combinatorios. Se puede considerar como una extensión del movimiento individual. Es similar con algunas otras implementaciones de un operador de vecindad, se puede ver el algoritmo de doble movimiento en el Cuadro 2.4.

El tamaño de la vecindad es

$$N(s) = \frac{E(E-1)}{2} \tag{2.4}$$

Donde E es el número de eventos. En algunos casos el número de timeslots es mucho más pequeño que el número de eventos. Por lo que esta vecindad puede resultar mayor que la primera. Entonces, se esperaría obtener mejores resultados con este tipo

1.Ejecutar un movimiento individual con el evento e_i (Cuadro 2.2)

1.1 Si el evento e_i se asignó, seleccione aleatoriamente otro evento e_j (*j* diferente a *i*) del mismo timeslot

1.2 Si el evento e_i no se asigna, seleccionar aleatoriamente otro evento e_j (*j* diferente de *i*)

2.Programe el evento e_j en su "mejor" timeslot

Obs: "mejor" en el sentido de la función de costo.

Cuadro 2.3: Movimiento individual con mejora heurística

1.Seleccionar aleatoriamente dos timeslots y tomar un evento cualquiera de cada timeslot (e_i diferente a e_j)

2.Intercambiar de timeslot entre los dos eventos

3.Si uno de los dos eventos no es programado, escoger el "mejor" timeslot para el evento no programado

Obs: "mejor" en el sentido de la función de costo

Cuadro 2.4: Algoritmo de doble movimiento

de movimiento.

Sin embargo, para problemas reales, el tamaño de esta vecindad es mucho más grande, por ejemplo, si hay 1,000 eventos, habrá 499,500 vecinos para cada solución. Ahora bien, se sabe que solo un fragmento de este número de vecinos podrá actualmente mejorar la calidad de la solución. Normalmente, la permutación de timeslots entre eventos de asignaturas diferentes deteriora la calidad de la solución [Melício et al., 2003].

Así pues, si limitamos los intercambios para eventos que pertenecen a la misma asignatura el número de vecinos sería.

$$N_c(s) = \sum_{c=1}^{|C|} \frac{E_c(E_c - 1)}{2} \tag{2.5}$$

Donde E_c es el número de eventos que pertenecen a la asignatura C. En el ejemplo anterior, si tenemos 50 asignaturas y cada asignatura tiene 20 eventos (es decir: cada asignatura se compone de 20 sesiones), lo que hace un total de 1000 eventos (mismo número que arriba). El tamaño de esta vecindad reducida sería de 9,500 vecinos.

El número total de eventos estará dado por,

$$E = \sum_{c=1}^{|C|} E_c \tag{2.6}$$

Y si cada asignatura tiene el mismo número de eventos, la expresión 2.6 vendrá a ser $E = |C| * E_c$. Este radio entre estas dos vecindades será igual a,

$$\frac{N(s)}{N_c(s)} = \frac{(|C| * E_c - 1)}{(E_c - 1)} = |C| + \frac{|C| - 1}{E_c - 1} \approx |C| \left(1 + \frac{1}{E_c}\right) \approx |C| \tag{2.7}$$

Donde $|N(s)|$ es el tamaño de la vecindad asociada con el doble movimiento y $|N_c(s)|$ es el tamaño de la vecindad relacionada con el *doble movimiento entre asignaturas*. Como se puede ver por la ecuación 2.7 el rango entre los tamaños de estas dos vecindades es aproximadamente proporcional al número de asignaturas.

2.4. Función de Costo

La función de costo juega un papel clave en cualquier problema de optimización. Es a través de su cálculo que uno puede medir la calidad de cualquier solución. La definición correcta de la función de costo es esencial para el desempeño de cualquier algoritmo de búsqueda. La función de costo esta dada por la siguiente expresión:

$$f(s) = \sum_{k} w_k C_k \tag{2.8}$$

Donde $s \in S$ es una solución en el espacio de soluciones S (puede ser una solución parcial) y los valores C_k representan cada uno de los objetivos que se tratan de optimizar, cargados con pesos por un factor w_k. Este peso representa la importancia relativa de la restricción relacionada. El objetivo de cualquier algoritmo de búsqueda sera encontrar una solución óptima $s*$ que minimice $f(s)$.

Por ejemplo, supongamos que se define una restricción C_0: que represente la suma en timeslots de todos los eventos no programados. El peso que afectaría esta restricción, el usuario no debería poder modificarla y se calcularía como sigue:

$$w_0 = \sum_{k=1}^{K} p_k w_k \tag{2.9}$$

Donde K es el número de restricciones definidas para el problema específico y p_k es el valor máximo que alcanzaría la violación de la restricción k si un evento se programa en un timeslot dado. Para todos los operadores de movimiento antes definidos en 2.3, la función de costo se calcula incrementalmente, es decir, solo se calcula el cambio en costo entre la nueva solución y la anterior.

2.5. Planteamiento del UCTP

Resolver el Problema de la Programación de Cursos en una Universidad (UCTP por sus siglas en Inglés de: University Course Timetabling Problem), al igual que el Problema General de Asignación de Recursos, no es sencillo debido a las restricciones que se consideran en el problema, las características de los recursos limitados y también la conside-

ración del tiempo y espacio para asignar los eventos. Todos estos elementos, hacen que el UCTP sea un problema de tipo NP-completo [Burke and Carter, 1997, Schaerf, 1999].

La finalidad al tratar el problema del UCTP es programar los cursos a nivel universitario para obtener la tabla de horarios semanal, de forma que todos los estudiantes puedan atender todas sus asignaturas programadas sin dificultades de horarios, por ejemplo: que el estudiante no tenga que atender dos eventos al mismo tiempo (llamada también: un choque de estudiante). En adición con la presente restricción de choque de estudiante, hay en la literatura modelos más complejos para crear las tablas de horarios, la mayor complejidad se presenta al añadir restricciones que hacen más compleja la obtención de soluciones, como por ejemplo, la disponibilidad de salones. También podría ser, al añadir restricciones que incluyen el fundamento de una función objetivo para evaluar la calidad de la solución más allá de la simple factibilidad [Carter et al., 1994].

Las restricciones para este tipo de problema se clasifican usualmente como restricciones duras y restricciones suaves. Las restricciones duras, son restricciones que deberán cumplirse y no deberán violarse bajo ninguna circunstancia, por ejemplo: un estudiante no puede atender dos clases al mismo tiempo. Las restricciones suaves, son restricciones que preferentemente se deben satisfacer; pero se puede aceptar la ocurrencia de algunas violaciones, desde luego, con una penalización asociada. Un ejemplo de este tipo de restricción podría ser: los estudiantes no deberán tener más de dos clases seguidas. Por razones pedagógicas y humanas, este tipo de restricción considera que, después de dos clases continuas, es necesario que el estudiante tenga algún descanso antes de tomar una tercera clase de forma continua; sin embargo, dadas las circunstancias del horario y asignación de clases, se tendrá que aceptar que algún o algunos estudiantes tengan que tomar más de dos clases seguidas. Entonces, con estas violaciones de la restricciones suaves, se tendrá que penalizar la función objetivo.

2.5.1. Descripción del UCTP

El UCTP planteado por Paechter [Rossi-Doria et al., 2003], que es una reducción de un problema real de Asignación de Recursos, consiste de un conjunto de eventos o clases

que se programan en un conjunto de 45 intervalos de tiempo (5 días a la semana, con 9 periodos de tiempo al día), un conjunto de salones donde se desarrollan los eventos, un conjunto de estudiantes que atienden los eventos, y un conjunto de facilidades que proporcionan los salones y que son necesarios para que se den los eventos. Cada estudiante atiende un número de eventos y cada salón tiene una capacidad. Entonces, una solución factible será aquella en la cual, todos y cada uno de los eventos, se programan para que se desarrollen en un periodo de tiempo y en un salón. Todo esto, de forma que se cumplan las siguientes restricciones duras:

- Un estudiante no atenderá más de un evento a la vez
- El salón tendrá el aforo suficiente para atender a todos los estudiantes del evento y satisfacer todas las necesidades requeridas por el evento
- En un salón solo habrá un evento a la vez

Además, una solución factible candidata se penalizará igualmente por cada ocurrencia en la violación de las siguientes restricciones suaves:

- Si un estudiante tiene una clase en la última hora del día
- Si un estudiante tiene más de dos clases seguidas
- Si un estudiante tiene una sola clase al día

El objetivo del problema es obtener una solución factible con el mínimo número de violaciones a las restricciones suaves. Cualquier solución infactible se considera mala solución.

2.5.2. Recursos y restricciones

Según el texto del UCTP, se identifican los siguientes conjuntos y elementos, formados por los recursos humanos y materiales:

- Un conjunto T de 45 *timeslots* (5 días por 9 periodos).
- Un conjunto de *salones* R en los cuales se imparten los eventos (cursos)

- Un conjunto de *estudiantes* A o alumnos que atienden los eventos
- Un conjunto de *facilidades* F. Cada salón proporciona un subconjunto de facilidades y cada evento necesita un subconjunto de facilidades.
- Cada *estudiante* s atiende un número de eventos
- Cada *salón* r tiene un aforo

Los timeslots son periodos de tiempo, donde se programarán los eventos (clases) y los estudiantes que asistirán al evento. Los 45 timeslots de cada salón se componen de 5 días de cursos a la semana con 9 periodos de tiempo cada día. Entonces, cada periodo de tiempo corresponde al lapso de tiempo en que se imparte una clase. Como se podrá percibir, la programación de 45 timeslots corresponderá a una Tabla de Horarios (Timetabling en Inglés).

Por facilidad, nos referiremos al término *periodo de tiempo* como *periodo* o directamente usaremos la palabra en Inglés *timeslot*.

Según establece el texto del problema de Paechter, una solución factible del UCTP, como se representa en la literatura, corresponderá a una Tabla de Horarios u Horario de Cursos, donde quedan programados todos y cada uno de los eventos que son asignados en los posibles 45 timeslots existentes para cada tabla. Esta asignación de eventos considera: El aforo del salón, las facilidades que ofrece el salón, las necesidades que requiere el evento y los estudiantes que asistirán al evento.

De la descripción del UCTP con las restricciones planteadas y los recursos que intervienen en el problema, identificamos que se tienen las siguientes restricciones duras y suaves:

Hay que recordar que para obtener una solución factible, todas y cada una de las restricciones duras se deberán satisfacer.

- $H1$: El salón debe satisfacer las necesidades del evento

- $H2$: Todos y cada uno de los eventos deberán ser programados
- $H3$: En un salón solo habrá un evento a la vez
- $H4$: El salón debe tener el aforo suficiente para atender a los estudiantes del evento
- $H5$: Un estudiante no atenderá mas de un evento a la vez

El texto original del problema UCTP, no considera la restricción $H2$; sin embargo nosotros la incluimos ya que es obvio que todos y cada uno de los eventos deberán ser programados y asignados.

Además, el UCTP candidato se penalizará igualmente por cada ocurrencia en la violación de las siguientes restricciones suaves:

- $S1$: Un estudiante no debe tener clase en el último periodo del día
- $S2$: Un estudiante no debe tener más de dos clases seguidas
- $S3$: Un estudiante no debe tener una sola clase al día

Al hacer un análisis del alcance de las restricciones suaves para que se puedan cumplir se puede inferir que: la restricción suave $S1$ se podrá verificar sin conocer el resto de la tabla de horarios; la restricción suave $S2$ es posible que se pueda verificar mientras se construye la solución; la restricción $S3$ se podrá verificar sólo cuando la tabla de horarios se haya completado y a todos los eventos (o clases) se les haya asignado un timeslot.

El objetivo del problema es obtener una solución factible con el mínimo número de violaciones de restricciones suaves. Todas las soluciones infactibles se consideran despreciables.

2.6. Representación Simbólica

A continuación se exhibe una de las representaciones del UCTP más conocidas que se encuentran en la literatura y que también es de las más utilizadas por los investigadores del área para utilizarla como base para realizar sus aplicaciones y presentar soluciones. La representación es una realización del grupo Metaheuristic Network

[Rossi-Doria et al., 2003] los cuales, utilizan estructuras interesantes para manejar, almacenar, procesar y presentar los datos y la solución.

Antes de hacer la programación de horarios (o tabla de horarios) el grupo Metaheuristic Network hace un *preprocesamiento* para designar a cada evento los posibles salones donde se podrá efectuar ese evento considerando las características y tamaño de los salones debido a las necesidades de cada evento.

La asignación de salones no forma parte de la representación explícita; en vez de ello utilizan un algoritmo de emparejamiento para generar la lista de posibles salones para cada evento. Entonces, para cada timeslot habrá una lista de eventos que tendrán lugar en él -eventos que se darán en ese horario- y una lista preprocesada de los posibles salones en los cuales estos eventos pueden ser asignados de acuerdo al tamaño y características. El algoritmo de emparejamiento da una máxima cardinalidad de emparejamiento entre estos dos conjuntos -la lista de eventos por timeslot y la lista de posibles salones para cada evento- usando un algoritmo determinístico de flujo. Si hubiera aun eventos no programados, los van tomando en orden secuencial y asignan cada uno en un salón que cumpla las características requeridas y el aforo adecuado, eligiendo primero los salones con menos eventos asignados. Si dos o más salones están parejos en la selección, toman aquel que aparezca primero en la lista -asignación por orden de aparición-. El grupo Metaheuristic Network indica que este procedimiento garantiza que cada asignación de evento-timeslot corresponde únicamente a una tabla de horarios, es decir una asignación completa de timeslots y salones para todos los eventos.

La representación de la solución que el grupo Metaheuristic Network utiliza es una representación directa de la solución. Donde una solución consiste de una lista ordenada de longitud $|E|$, las posiciones de la lista corresponden a los eventos (la posición i corresponde al evento i para $i = 1, \ldots, |E|$). Un número entero entre 1 y 45 (representa un timeslot) colocado en la posición i indicará el timeslot donde se programó el evento i. Por ejemplo la lista

1	2	3	$\ldots$	$\lvert E\rvert$	$\leftarrow$	Cantidad de eventos (de la lista ordenada)
3	27	43	$\ldots$	10	$\leftarrow$	Nos. entre 1 y 45, indican el timeslot (horario) en que se programaron los eventos i

Ejemplo: El evento 1 se programó en el timeslot 3; el evento 2 esta programado en el timeslot 27; el evento 3 se programó en el timeslot 43, etc.

Como hay un tiempo límite para encontrar las soluciones, junto con el concepto de un algoritmo potente, se necesitan estructuras de datos eficientes y representaciones internas del problema para obtener resultados competitivos.

2.7. La Estructura de Vecindad y Búsqueda Local

Los algoritmos tradicionales de búsqueda local parten de una solución inicial que de modo paulatino es transformada en otras que a su vez son mejoradas al introducirles movimientos sencillos. Los movimientos más utilizados son la reubicación de timeslot de algún evento o el intercambio de eventos entre dos timeslots. Si el cambio da lugar a una mejor solución que la actual, se sustituye ésta por la nueva, continuando el proceso hasta que no es posible ninguna nueva mejora o hasta alcanzar un determinado número de ciclos de computación. De esta forma de búsqueda local tradicional existen ligeras variaciones y el proceso de búsqueda se realiza computacionalmente de acuerdo a la estructura de vecindad que se implementa en un algoritmo.

A continuación se resume el funcionamiento de la estructura de vecindad y búsqueda local que ponen en práctica el grupo Metaheuristic Network [Rossi-Doria et al., 2003].

La representación de la solución antes descrita permite definir una vecindad utilizando movimientos sencillos que solo involucran timeslots y eventos. Recordar que la asignación de salones son responsabilidad del algoritmo de emparejamiento.

La Estructura de Vecindad es la unión de dos vecindades más pequeñas, N_1 definida

por un operador que mueve un solo evento a un timeslot diferente, y N_2 definida por un segundo operador que permuta dos eventos de sus timeslots.

El procedimiento en la aplicaciones de los movimientos es que, para el evento que este siendo considerado, los movimientos potenciales se procesan en estricto orden. Primero se aplica el operador N_1, que trata de mover el evento al siguiente timeslot, luego al siguiente, luego al siguiente etc, siempre considerando que se pueda asignar un evento donde se obtenga una mejor solución. Si esta búsqueda a través de N_1 falla, es decir que después de un número premeditado de búsquedas no encuentra un mejor timeslot que mejore la solución, entonces se aplica el operador N_2 que trata de permutar con el evento siguiente de la lista, luego con el siguiente, con la misma filosofía de encontrar mejor solución, y así sucesivamente. En cada movimiento individual, se requiere hacer cálculos para conocer la magnitud de la función de costo. El funcionamiento del tipo de búsqueda local así aplicada es tipo ”Greedy”, ya que la búsqueda se hace en estricto orden, no se considera optimizar la búsqueda ni búsquedas focalizadas, por lo tanto, busca en todo el espacio de forma dirigida y para encontrar una solución factible, con cada evento que no cumpla restricciones duras puede llegar a recorrer hasta 2 veces todo el espacio de búsqueda. Por esta forma de búsqueda, este tipo de búsqueda toma demasiado tiempo.

El grupo Metaheuristic Network en su artículo [Rossi-Doria et al., 2002] mencionan que en la búsqueda local que implementaron para el UCTP, aparte de incluir los movimientos N_1 y N_2, aplican un tercer movimiento N_3 donde intervienen 3 eventos y que al parecer, su funcionamiento es igual al N_2. Sin embargo, en la literatura se encuentra que la mayoría de autores utilizan como base de su desarrollo los movimientos N_1 y N_2 y a partir de ellos, hacen variaciones o incluyen algún tipo de movimiento particular según su estrategia de análisis y aplicación.

La Búsqueda Local es una primera mejora estocástica de búsqueda local basada en la estructura de vecindad descrita. La búsqueda local es una acción repetitiva de prueba y error con cada evento a partir de la lista que contiene todos los eventos. De la lista hace una selección de cada evento en orden aleatorio, y para cada evento involucrado

en la violación de restricciones, trata todos los posibles movimientos en la vecindad hasta encontrar una mejora. Primero resuelve la violación de restricciones duras, y luego, si se alcanza la factibilidad, también resuelve la violación de restricciones suaves. Se aplica a las soluciones, una evaluación que el grupo Metaheuristic Network le llama Delta para permitir una búsqueda rápida a través de las soluciones vecinas (cercanas). La descripción detallada de esta búsqueda local se muestra en el Algoritmo 2.1.

Como la búsqueda local puede tomar una cantidad considerable de tiempo de CPU, podría ser más efectivo, dentro del contexto de algunas de las metaheurísticas, usar este tiempo en forma diferente. Por ejemplo, introducir en la búsqueda local un parámetro para controlar el número máximo de pasos permitidos, el cual dejan libre en las implementaciones de las diferentes metaheurísticas.

Algoritmo 2.1 Búsqueda Local

1. Ev-count $\leftarrow 0$;
   Generate a circular randomly-ordered list of the events;
   Initialize a pointer to the left of the first event in the list;

2. Move the pointer to the next event;
   Ev-count $\leftarrow$ Ev-count + 1;
   If (Ev-count $= |E|$) $\{$
   Ev-count $\leftarrow 0$;
   goto 3.;
   $\}$

   a) if (current event NOT involved in hard constraint violation (hcv)) goto 2.;

   b) if ($\nexists$ an untried move for this event) goto 2.;

   c) Calculate next move (first in N_1, then N_2) and generate resulting potential timetable;

   d) Apply the matching algorithm to the timeslots affected by the move and delta-evaluate the result;

   e) if (move reduces hcvs) $\{$
   Make the move;
   Ev-count $\leftarrow 0$;
   goto to 2.;
   $\}$
   else goto 2.b;

3. if ($\exists$ any hcv remaining) END LOCAL SEARCH;

4. Move the pointer to the next event;
   Ev-count $\leftarrow$ Ev-count + 1;
   if (Ev-count $= |E|$) END LOCAL SEARCH;

   a) if (current event NOT involved in soft constraint violation (scv)) goto 4.;

   b) if ($\nexists$ an untried move for this event) goto 4.;

   c) Calculate next move (first in N_1 then N_2) and generate resulting potential timetable;

   d) Apply the matching algorithm to the timeslots affected by the move and delta-evaluate the result;

   e) if (move reduces scvs without introducing a hcv) $\{$
   Make the move;
   Ev-count $\leftarrow 0$;
   goto to 4.;
   $\}$
   else goto 4.b;

Capítulo 3

Modelo Matemático del UCTP

En este capítulo se desarrollará la formulación del UCTP para obtener el modelo matemático. La formulación deberá considerar los recursos materiales y humanos, las restricciones duras y suaves para hacer una representación correcta del problema. Primero se presentará la formulación del modelo matemático y posteriormente se dará una explicación detallada de que representan cada una de las fórmulas que componen el modelo matemático.

La formulación del Modelo Matemático que a continuación se presenta, es una de las aportaciones que se planeó desarrollar e incluir en este trabajo doctoral.

3.1. Formulación

A partir del texto del UCTP, planteado por Paechter [Rossi-Doria et al., 2003] y que mostramos en el Capítulo 2.5.1, una solución factible es aquella donde se programan todos y cada uno de los eventos asignándolos en los posibles 45 timeslots existentes para cada salón. Esta programación de eventos considera: El aforo del salón, las facilidades que ofrece el salón, las necesidades que requiere el evento y los estudiantes que asistirán al evento. Todo esto, de forma que las restricciones duras ($H1$ a $H5$) se deberán satisfacer. Además, la solución candidata se penalizará igualmente por cada ocurrencia en la violación de las restricciones suaves ($S1$ a $S3$).

Entonces, sean los conjuntos:

- De eventos: $E = \{1, 2, 3, \ldots, n_E\}$
- De timeslots: $T = \{1, 2, 3, \ldots, 45\}$
- De días: $D = \{1, 2, 3, 4, 5\}$
- De periodos: $P = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- De salones: $R = \{1, 2, 3, \ldots, n_R\}$
- De estudiantes: $A = \{1, 2, 3, \ldots, n_A\}$
- De facilidades: $F = \{1, 2, 3, \ldots, n_F\}$

Las restricciones duras que se deberán satisfacer:

- $H1$: Un salón debe satisfacer las necesidades de los eventos que allí se programen
- $H2$: Todos y cada uno de los eventos deberán ser programados
- $H3$: En un salón solo habrá un evento a la vez
- $H4$: El salón debe tener el aforo suficiente para atender a los estudiantes del evento
- $H5$: Un estudiante no atenderá mas de un evento a la vez

Las restricciones suaves:

- $S1$: Un estudiante no debe tener clase en el último periodo del día
- $S2$: Un estudiante no debe tener más de dos clases seguidas
- $S3$: Un estudiante no debe tener una sola clase al día

A partir del conjunto $T = \{1, 2, 3, \ldots, 45\}$ de timeslots que representa 45 periodos disponibles en una semana escolar de 5 días con 9 periodos por día. Se desea programar los eventos y los estudiantes, y encontrar la Tabla de Horarios para cada salón. Cada

salón puede ofrecer diferentes facilidades y cada evento puede tener diferentes necesidades.

A continuación se presenta la formulación matemática que representa el modelo del UCTP:

Encontrar una solución satisfactoria para el UCTP donde se cumpla que la Función Objetivo (FO) sea mínima.

$$Minimizar\ FO = \sum_{a=1}^{n_A} F1(a, S1) + \sum_{a=1}^{n_A} F2(a, S2) + \sum_{a=1}^{n_A} F3(a, S3) \tag{3.1}$$

Sujeta a:

$$X(e, \Phi(t,r)) = \begin{cases} 1, \text{ si } \Phi(t,r) = 1 \text{ y } R(r,e) = 1 \\ 0, \text{ de otra forma} \end{cases} \quad \forall e \in E \tag{3.2}$$

$$\sum_{t=1}^{45} \sum_{r=1}^{n_R} X(e, \Phi(t,r)) \geq 1 \qquad \forall e \in E \tag{3.3}$$

$$\sum_{e=1}^{n_E} X(e, \Phi(t,r)) \leq 1 \qquad \forall t \in T, r \in R \tag{3.4}$$

$$\sum_{a=1}^{n_A} Z(a, X(e, \Phi(t,r))) \leq Aforo(r) \quad \forall e \in E, t \in T, r \in R \tag{3.5}$$

$$\sum_{e=1}^{n_E} \sum_{r=1}^{n_R} Z(a, X(e, \Phi(t,r))) \leq 1 \qquad \forall a \in A, t \in T \tag{3.6}$$

Para $t, r, e, a \in (\mathcal{N} - \{0\})$

En esta formulación, la expresión 3.1, que indica minimizar la Función Objetivo, contabiliza exclusivamente las violaciones que se hayan cometido de las restricciones suaves $S1$, $S2$ y $S3$ que están relacionadas con la variables a que representa a los estudiantes, para esto, se hace un barrido de las funciones $F1$, $F2$ y $F3$ donde se establece la marcación de las violaciones respectivas. El detalle de estas funciones se muestra en las expresiones 3.16 a 3.18 del Capítulo 3.2. Por otro lado, las expresiones 3.2 a 3.6 representan las restricciones duras $H1$ a $H5$ que es obligatorio cumplir. En el Capítulo 3.2

se da una explicación detallada de todas las expresiones de esta formulación matemática.

La relación de los conjuntos D de días y P de periodos forman el conjunto de 45 timeslots, que al hacer la programación de eventos y asignarlos en los timeslots, se obtendrá un Itinerario o Tabla de Horarios para cada salón. Entonces, la solución que se obtenga tendrá tantas tablas como salones R existan. Por lo tanto, una solución estará formada por el conjunto de Tablas de Horarios. En el Cuadro 3.1 se muestra un ejemplo de Tabla de Horarios para un salón r.

Salón r	Días D				
Periodos P	1	2	3	4	5
1	*1*	*2*	*3*	*4*	*5*
2	*6*	*7*	*8*	*9*	*10*
3	*11*	*12*	*13*	*14*	*15*
4	*16*	*17*	*18*	*19*	*20*
5	*21*	*22 (5, 2)*	*23*	*24*	*25*
6	*26*	*27*	*28*	*29*	*30*
7	*31*	*32*	*33*	*34*	*35*
8	*36*	*37*	*38*	*39*	*40*
9	*41*	*42*	*43*	*44*	*45*

Cuadro 3.1: Ejemplo. Tabla de Horarios de un salón r

La Tabla de Horarios es la abstracción de una estructura bidimensional para cada salón r con la programación de horarios de la semana, que se forma con los días D en el eje x y los periodos P en el eje y. En esta estructura bidimensional se programarán los eventos que se impartirán y los estudiantes que atenderán cada uno de esos eventos durante la semana. Además, el salón deberá ofrecer las facilidades necesarias para que se efectúe el evento. Al hacer la programación de eventos y estudiantes en los diferentes salones, la variable R (de salones) crea una tercera dimensión (eje z) para formar el conjunto de las Tablas de Horarios. Por lo tanto, con el conjunto de salones se obtiene una estructura tridimensional con dimensiones D, P y R que al tener asignados los estudiantes y los eventos, esta abstracción de estructura tridimensional forma la solución del UCTP.

3.2. Detalle de la formulación

A continuación hacemos una descripción más detallada de la formulación del Modelo Matemático obtenido en el Capítulo 3.1.

Como se expuso, una solución factible es aquella donde se programan todos y cada uno de los eventos asignándolos en los posibles 45 timeslots existentes para cada salón. Esta asignación de eventos considera: el aforo del salón, las facilidades que ofrece el salón, las necesidades que requiere el evento y los estudiantes que asistirán al evento. Todo esto, de forma que las restricciones duras ($H1$ a $H5$) se satisfagan. Además, la solución candidata se penalizará igualmente por cada ocurrencia en la violación de las restricciones suaves ($S1$ a $S3$).

A partir de los recursos humanos y materiales que se quiere programar, se tienen los siguientes conjuntos:

- De eventos: $E = \{1, 2, 3, \ldots, n_E\}$
- De timeslots: $T = \{1, 2, 3, \ldots, 45\}$
- De días: $D = \{1, 2, 3, 4, 5\}$
- De periodos: $P = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- De salones: $R = \{1, 2, 3, \ldots, n_R\}$
- De estudiantes: $A = \{1, 2, 3, \ldots, n_A\}$
- De facilidades: $F = \{1, 2, 3, \ldots, n_F\}$

Se desea encontrar una solución satisfactoria para el UCTP donde se cumpla que la Función Objetivo (FO) sea mínima:

$$Minimizar\ FO = \sum_{a=1}^{n_A} F1(a, S1) + \sum_{a=1}^{n_A} F2(a, S2) + \sum_{a=1}^{n_A} F3(a, S3) \tag{3.7}$$

Sujeta a que se deban cumplir las restricciones duras $H1$, $H2$, $H3$, $H4$, $H5$ (enunciadas en el Capítulo 3.1) ***y sancionar cada ocurrencia de violación de las restricciones suaves*** $S1$, $S2$ y $S3$ (también, enunciadas en el Capítulo 3.1).

A partir del conjunto $T = \{1, 2, 3, \ldots, 45\}$ de timeslots que representa los 45 lugares disponibles en una semana escolar para un salón y que forman una Tabla de Horarios de 5 días con 9 periodos por día, donde se programarán los eventos y los estudiantes.

Se define un par (t, r), donde $t \in T$ representa un timeslot y $r \in R$ representa un salón. Entonces, para cada (t, r):

$$\Phi(t, r) = \begin{cases} 1, \text{ si se asigna el timeslot } t \text{ del salón } r \\ 0, \text{ de otra forma} \end{cases} \tag{3.8}$$

Por ejemplo. Si $\Phi(22, 3) = 1$ indicará que el timeslot 22 del salón 3 ya está asignado. Esta representación se ilustra en el Cuadro 3.1, donde se considera que el salón r es igual a 3 y el timeslot 22 (con coordenadas $p = 5$ y $d = 2$) está ocupado.

Cumplimiento de la restricción $H1$: *Un salón debe satisfacer las necesidades de los eventos que allí se programen*.

Para cada salón $r \in R$ y para cada evento $e \in E$ se define:

$$R(r, e) = \begin{cases} 1, \text{ si el salón } r \text{ satisface las necesidades del evento } e \\ 0, \text{ de otra forma} \end{cases} \tag{3.9}$$

Además, para cada evento $e \in E$, timeslot $t \in T$ y salón $r \in R$ se define:

$$X(e, \Phi(t, r)) = \begin{cases} 1, \text{ si } \Phi(t, r) = 1 \text{ y } R(r, e) = 1 \\ 0, \text{ de otra forma} \end{cases} \tag{3.10}$$

La expresión 3.10 será verdadera si las expresiones 3.8 y 3.9 son verdaderas. Esto indica que el salón r, satisface las necesidades del evento e que se da en el timeslot t.

Por lo tanto, las expresiones 3.9 y 3.10 satisfacen la restricción $H1$.

Cumplimiento de la restricción $H2$: *Todos y cada uno de los eventos deberán ser programados*.

Como a cada evento e se le tiene que asignar un espacio y un horario para efectuar el evento, entonces para cada $e \in E$

$$\sum_{t=1}^{45} \sum_{r=1}^{n_R} X(e, \Phi(t, r)) \geq 1 \tag{3.11}$$

La expresión 3.11 checa que todo evento e este programado en algún timeslot t de algún salón r. Además, el evento puede estar programado varias veces en la semana.

Por lo tanto, la expresión 3.11 cumple la restricción $H2$.

Cumplimiento de la restricción $H3$: *En un salón solo habrá un evento a la vez*.

Si para algún timeslot $t \in T$ y algún salón $r \in R$, $X(e, \Phi(t, r)) = 1$. Es decir, si el evento e se programa en el timeslot t del salón r. Entonces, ningún otro evento e' podrá programarse en el timeslot t del salón r.

Así que: $X(e', \Phi(t, r)) = 0 \ \ \forall \, e' \in E$, con $e' \neq e$. Entonces,

$$\sum_{e=1}^{n_E} X(e, \Phi(t, r)) \leq 1 \tag{3.12}$$

Por lo tanto, la expresión 3.12 satisface la restricción $H3$.

Cumplimiento de la restricción $H4$: *El salón debe tener el aforo suficiente para atender los estudiantes del evento*.

Para cada estudiante $a \in A$ se define:

$$Z(a, X(e, \Phi(t, r))) = \begin{cases} 1, \text{ si } X(e, \Phi(t, r)) = 1 & \text{Entonces, el estudiante} \\ & a \text{ atiende el evento } e \\ 0, \text{ de otra forma} & \end{cases} \tag{3.13}$$

La expresión 3.13 será verdadera, si el estudiante a atiende el evento e que se da en el timeslot t del salón r.

Entonces, para cada salón r se deberá cumplir que:

$$\sum_{a=1}^{n_A} Z(a, X(e, \Phi(t, r))) \leq Aforo(r) \tag{3.14}$$

La expresión 3.14 indica que la suma de estudiantes que atienden el evento e no debe rebasar el aforo del salón r.

Por lo tanto, la expresión 3.14 satisface la restricción H4.

Cumplimiento de la restricción $H5$: *Un estudiante no atenderá mas de un evento a la vez*.

Para algún estudiante $a \in A$, si la expresión 3.13 es verdadera. Es decir, que el estudiante a se encuentra atendiendo el evento e en el salón r. Entonces, este estudiante no podrá atender otro evento e' al mismo tiempo.

Así que: $Z(a, X(e', \Phi(t, r))) = 0$, $\forall e' \in E$, con $e' \neq e$. Entonces:

$$\sum_{e=1}^{n_E} \sum_{r=1}^{n_R} Z(a, X(e, \Phi(t, r))) \leq 1 \tag{3.15}$$

La expresión 3.15 indica que si la expresión 3.13 es verdadera. Entonces, el estudiante a no podrá encontrarse atendiendo otro evento.

Por lo tanto, la expresión 3.15 satisface la restricción $H5$.

A continuación se da el detalle de la formulación de las restricciones suaves, para ello se utilizan las funciones $F1$, $F2$ y $F3$ que se integran a la ecuación 3.7 con el fin de contabilizar la ocurrencia de violaciones y obtener la calidad de la función objetivo:

Para sancionar la ocurrencia de la Restricción $S1$: *Un estudiante no debe tener clase en el último periodo del día*.

Sea el timeslot t, algún evento $e \in E$ y algún salón $r \in R$. Tenemos, que para cada estudiante $a \in A$ se define:

$$F1(a, S1) = \begin{cases} 1, \text{ si } Z(a, X(e, \Phi(t, r))) = 1 \text{ para } t \in 41 \leq t \leq 45 \\ 0, \text{ de otra forma} \end{cases} \tag{3.16}$$

Para sancionar la ocurrencia de la Restricción $S2$: *Un estudiante no debe tener más de dos clases seguidas*.

Sea un día $d \in D$, para cada estudiante $a \in A$ se define $Ad = \{5p + d \mid 0 \leq p \leq 8\}$ y para cada subconjunto $\{a1, a2, a3\}$ de tres elementos del conjunto Ad, tal que $a1 = 5p_1 + d$, $a2 = a1 + 5$ y $a3 = a1 + 10$.

Entonces, para algún salón $r \in R$, algún evento $e \in E$ y algún p_1 con $0 \leq p_1 \leq 6$, se define:

$$F2(a, S2) = \begin{cases} 1, \text{ si } Z(a, X(e, \Phi(a, r))) = 1 \text{ para cada } a \in \{a1, a2, a3\} \\ 0, \text{ de otra forma} \end{cases} \tag{3.17}$$

Para sancionar la ocurrencia de la restricción $S3$: *Un estudiante no debe tener una sola clase al día*.

Sea un día $d \in D$, para cada estudiante $a \in A$, se define:

$$F3(a, S3) = \begin{cases} 1, \; si \sum_{r=1}^{n_R} \sum_{e=1}^{n_E} \sum_{p=0}^{8} Z(a, X(e, \Phi(5p + d, r))) = 1 \\ 0, \text{ de otra forma} \end{cases} \tag{3.18}$$

Capítulo 4

Métodos para resolver el UCTP

Resolver un problema combinatorio se puede resumir en encontrar una solución que sea *la mejor* o *la óptima* dentro de un conjunto de posibles alternativas.

En general los problemas que trata la optimización combinatoria se consideran difíciles de resolver y muchos de ellos inclusive caen en los problemas NP-duros, esto es, el tiempo que requieren para resolver una instancia de ese problema crece en el peor de los casos de manera exponencial con respecto al tamaño de la instancia. Por lo mismo, muchas veces se tienen que resolver usando métodos de solución aproximada, que dan buenas soluciones, inclusive a veces soluciones óptimas, a bajo costo computacional.

Para resolver el Problema de la Programación de Cursos en una Universidad se han desarrollado varias técnicas, como las metaheurísticas. Estas metaheurísticas son algoritmos que se pueden utilizar como métodos determinísticos o estocásticos [Schaerf, 1999].

Las metaheurísticas se pueden usar para resolver los problemas de optimización, ya que siendo estos problemas de alto grado de dificultad, el tratar de obtener una solución de forma determinística podría tomar tiempos de computación demasiado elevados o definitivamente no la encuentran con instancias consideradas grandes, ya que tienen que hacer una exploración intensiva en el espacio de búsqueda, inclusive en muchos casos no alcanzaría la vida de algún humano el tratar de obtener soluciones directas con los recursos actuales de computación ya que la solución puede tener alto grado de infactibili-

dad. También, podrían usarse muchas de las estrategias de búsqueda conocidas, aunque algunas pueden resultar ser demasiado costosas computacionalmente o durante el proceso de búsqueda de la solución quedar atrapados en mínimos locales.

Algunas de las propiedades deseables para una metaheurística son:

- Simplicidad: debe de ser simple y basada en un principio claro que pueda ser aplicable en general.
- Precisión: debe de estar formulada en términos matemáticos precisos.
- Coherencia: todos los pasos deben de seguir en forma natural los principios de la metaheurística.
- Eficiencia: debe de tomar un tiempo razonable de tiempo.
- Efectividad: debe de encontrar las soluciones óptimas para la mayoría de los problemas de prueba en donde se conoce su solución.
- Robustez: debe ser consistente en una amplia variedad de instancias.
- Amigable: debe de ser claramente expresada, fácil de entender y fácil de usar, y con la menor cantidad de parámetros posibles.

Sin embargo, la mayoría de las metaheurísticas cumplen con solo algunas de las propiedades arriba mencionadas y no existe un claro ganador de cual es mejor o cual incluye más propiedades ya que en general dependen de la naturaleza de los problemas y los objetivos buscados.

La mayoría de los métodos en las metaheurísticas son: algoritmos basados en construcción o algoritmos basados en búsqueda local.

Los basados en construcción trabajan con soluciones parciales que tratan de completar, mientras los de búsqueda local se mueven en el espacio de soluciones completas.

Los algoritmos de construcción encuentran soluciones de manera incremental empezando con una solución vacía, añadiendo componentes de forma incremental hasta completar una solución. Generalmente los algoritmos que construyen soluciones son más rápidos, pero sus soluciones no siempre son buenas.

En la optimización de recursos, así como en la inteligencia artificial, el proceso de encontrar una solución es principalmente por medio de la búsqueda y evaluación. La búsqueda es necesaria en la solución de problemas y normalmente involucra introducir heurísticas. Las heurísticas son criterios, métodos, o principios para decidir cuál de varias alternativas de acción promete ser la más efectiva para cumplir con una meta. Representan un compromiso entre: (i) simplicidad y (ii) poder discriminatorio entre opciones buenas y malas.

En problemas complejos, las heurísticas juegan un papel fundamental para reducir el número de evaluaciones y para obtener soluciones dentro de restricciones de tiempo razonables.

La mayoría de los problemas se pueden plantear tanto como de optimización como de satisfacción de restricciones. La diferencia entre estas dos formas puede ser muy importante. Por ejemplo: encontrar una ruta entre ciudades en el problema del agente viajero es trivial, encontrar la ruta más corta es NP.

Una idea fundamental dentro de los métodos heurísticos es que lo más *corto/barato* es lo más *rápido/mejor*.

Si existe un criterio de aceptación (tolerancia) se habla de un problema de semi-optimización. Cuando se habla de alta probabilidad, se tiene un problema de optimización aproximada.

La mayoría de los problemas son de *semi-optimización*, estableciendo un balance razonable entre calidad de la solución y costo de solución.

Algo un poco más ambicioso es dotar a los algoritmos con parámetros ajustables para cambiar el compromiso entre calidad y costo.

En muchos casos es deseable incluir, dentro de la representación, información adicional que nos defina el subproblema que falta por resolver. El código que especifica ésta información adicional se llama un estado.

En una representación basada en estados el proceso de solución trata de encontrar una secuencia de operaciones que transformen al estado inicial en uno final.

En general, si la solución puede especificarse como un camino o un estado, la representación basada en estados es más adecuada.

Por otro lado si la solución es más conveniente representarla como un árbol, entonces una representación de reducción de problemas o grafo es más adecuada.

Algo que se tiene que considerar al escoger una metaheurística es considerar ¿cómo encontrar una buena heurística? ¿Cómo usarla para que sea más efectiva? y ¿Cómo evaluar sus méritos? De estas interrogantes, se podría escoger cual metaheurística podría resultar más adecuada para resolver un problema de optimización.

A continuación se presentan cinco de las metaheurísticas más utilizadas en la literatura para tratar de obtener soluciones factibles del UCTP.

4.1. Optimización con Colonia de Hormigas

La optimización con colonia de hormigas (ant colony optimization o ACO) está inspirado en el rastro y seguimiento de feromonas que realizan las hormigas como medio de comunicación. ACO es una metaheurística propuesta por Dorigo y otros [Dorigo et al., 1996] que se inspira en el comportamiento de las hormigas en la búsqueda de comida para su

colonia.

Las hormigas se mueven concurrentemente e independientemente. Durante la búsqueda de comida, cada hormiga va dejando un rastro de feromona que crece si la misma hormiga u otras hormigas pasan por el mismo camino. El rastro de feromona también esta sujeto a la evaporación durante el tiempo. Entonces, los caminos de feromonas sirven como información distribuida que las hormigas usan en forma probabílistica para construir soluciones a un problema y que las hormigas adaptan para reflejar su experiencia.

ACO es una estrategia de construcción, donde la solución se forma probabilísticamente al ir añadiendo componentes de soluciones parciales considerando: (i) heurísticas para resolver el problema particular y (ii) trazas de feromona.

Para la representación de un problema se requiere definir: (i) un conjunto de componentes y (ii) estados definidos en términos de secuencias de componentes (transiciones entre estados).

La metodología de ACO construye soluciones moviéndose en un grafo de construcción, donde los vértices son componentes del problema y los arcos conecciones entre estos componentes. Para construir una solución se sigue cierta política dada por las restricciones del problema.

Los componentes y las conecciones pueden tener asociadas cierta cantidad de feromona e información heurística acerca del problema.

Al añadir un componente en una solución parcial, se puede actualizar la cantidad de feromona.

También al llegar a una solución completa, se pueden ver todos los pasos que se siguieron y también actualizar los niveles de feromona del camino.

Además, en ACO se incluyen dos procedimientos adicionales: (1) la evaporación (pheromone trail evaporation) y (2) acciones tipo demonio (daemon actions).

El proceso de evaporación define como decrecer la cantidad de feromona en el tiempo.

Las acciones tipo demonio se pueden usar para tratar las acciones centrales/globales que no pueden lograrse con las hormigas individuales.

ACO se ha aplicado con éxito en varios problemas de optimización combinatoria, como el agente viajero y la programación de recursos como el UCTP entre otros. De hecho, la primera versión de ACO se utilizó para resolver el agente viajero, considerando que el agente viajero trata de resolver el camino más corto entre dos ciudades o nodos.

A continuación se muestra la formulación de la metaheurística ACO.

Al construir soluciones, una hormiga k decide irse de la ciudad i a la ciudad j con la siguiente probabilidad:

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}(t)]^\alpha \cdot [\eta_{il}]^\beta} \text{ si } j \in \mathcal{N}_i^k$$

donde $\eta_{ij} = 1/d_{ij}$ es información disponible (la visibilidad del rastro), la cual es inversamente proporcional a la distancia entre dos nodos. α y β son parámetros que se tienen que definir, y $\mathcal{N}_i^k$ es la vecindad factible de la hormiga k (el conjunto de ciudades que k no ha visitado).

Si $\alpha = 0$ se visita la ciudad más cercana. Si $\beta = 0$ se basa solo en las trazas de feromona y tiende a converger rápidamente a un punto de no mejora subóptimo.

Cuando todas las hormigas completan un circuito, se actualizan las feromonas. Primero se reducen todos los caminos por un factor constante (evaporación) y después cada hormiga deposita la siguiente cantidad de feromona en los nodos de su circuito:

$$\forall(i,j)\ \tau_{ij}(t+1) = (1-\rho)\cdot\tau_{ij}(t) + \sum_{k=1}^{m}\Delta\tau_{ij}^{k}(t)$$

donde $0 \leq \rho \leq 1$ es la razón de evaporación de feromona y m es el número de hormigas.

$\Delta\tau_{ij}^{k}(t)$ es la cantidad que deposita cada hormiga en cada nodo, definida por:

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} 1/L^{k}(t), \text{ si la liga } (i,j) \text{ es usada por la hormiga } k \\ 0, \text{ de otra forma} \end{cases}$$

donde $L^{k}(t)$ es la longitud del circuito de la hormiga k.

Se han diseñado algunas extensiones de ACO, variaciones que se han desarrollado según las necesidades o estrategias de los autores para resolver sus problemas.

Una primera extensión fue la de premiar de más a la mejor solución global obtenida hasta el momento, algo parecido a una estrategia elitista.

Otra variante fue ordenar las hormigas de mejor a peor y solo permitir a las N mejores poner feromonas (además de la mejor ruta global).

Otra variante fue el mover a las hormigas usando una política $\epsilon-$greedy.

Algunas variantes solo actualizan la cantidad de feromona del camino de la mejor hormiga. El uso de feromonas sirve para balancear entre exploración y explotación.

Un sistema puede utilizar un límite inferior y superior de cantidad de feromona. El inferior garantiza un nivel mínimo de exploración.

También se pueden variar los parámetros α y β para balancear de forma diferente la exploración y explotación durante la búsqueda.

Se puede combinar con búsqueda local (parecido a GRASP). Se construye con ACO y se sigue con búsqueda local.

Si existen muchos posibles candidatos en la parte de construcción de soluciones se puede tener también una lista de candidatos.

A continuación se muestra un algoritmo de Colonia de Hormigas que es adecuado para tratar el problema del UCTP:

$\tau(e,t) \leftarrow \tau_0$, $\forall (e,t) \in E$xT
input: Una instancia problema I
calcula $c(e,e')$, $\forall (e,e') \in E^2$
calcula $d(e), f(e), s(e)$, $\forall e \in E$
ordena E de acuerdo a $\prec$, para obtener $e_1 \prec e_2 \prec \cdots \prec e_n$
$j \leftarrow 0$
while tiempo límite no se alcanza **do**
  $j \leftarrow j+1$
  for $a = 1$ **to** m **do**
    {construye el proceso de la hormiga a}
    $A_0 \leftarrow 0$
    for $i = 1$ **to** n **do**
      escoge aleatoriamente el timeslot t de acuerdo a la distribución
      de probabilidad P para el evento e_i
      ejecuta actualización de feromona local para $\tau(e_i,t)$
      $A_i \leftarrow A_{i-1} \cup (e_i,t)$
    end for
    $s \leftarrow$ la solución después de aplicar el algoritmo de
    emparejamiento para A_n
    $s \leftarrow$ la solución después de aplicar $h(j)$ pasos de
    búsqueda local para llegar a s
    $s_{best} \leftarrow$ la mejor solución de s y C_{best}
  end for

actualiza la feronoma global para $\tau(e,t) \forall (e,t) \in ExT$ usando C_{best}

end while

output: Una solución optimizada candidata s_{best} para I

En este algoritmo para hacer la programación, cada una de las m hormigas *construye/asigna*, evento por evento, hasta hacer una asignación completa de todos los eventos. Para hacer una sola asignación de un evento a un timeslot, una hormiga toma el siguiente evento desde una lista preordenada-ordenada, y de forma probabilista escoge un timeslot para el evento, guiada por dos tipos de información: (i) la información heurística, la cual es una evaluación de las violaciones de restricciones causadas al hacer las asignaciones, y (ii) la información de la intensidad del aroma τ en la forma de un nivel de ”feromona”, el cual es una estimación de utilidad al hacer la asignación, según una cantidad determinada de iteraciones consideradas en el algoritmo. La información del nivel de feromona esta representado por una matriz que da el valor de “feromona” $\tau : E$x$T \rightarrow \mathbb{R}_{\geq 0}$, donde E es el conjunto de eventos y T es el conjunto de timeslots.

4.2. Búsqueda Local Iterada

La Búsqueda Local Iterada [Lourenço et al., 2002] está basada en la idea de mejorar un procedimiento de búsqueda local proporcionando nuevas soluciones de inicio que se obtienen de perturbaciones que se aplican a la solución actual, esto, a menudo conduce a resultados mucho mejores que cuando se usa reinicio al azar [Paquete and Stützle, 2002].

El problema de cualquier búsqueda local es que, durante la búsqueda de soluciones en la vecindad, se puede quedar atrapado en mínimos locales si no se implementa un control adecuado para salir del pozo. Sin embargo, la característica que tiene ILS de reiniciar la búsqueda desde varios puntos aleatorios no soluciona este problema.

En realidad lo que se necesita es que se haga un muestreo sesgado, guiando la búsqueda de forma más efectiva. En la literatura se pueden encontrar algunas ideas con las que han trabajado. Una de ellas es generar pozos de atracción vecinos (o al menos diferentes) donde se establece un camino de vecinos seleccionados aleatoriamente,

$s_1, s_2, \ldots, s_i$, donde s_{j+1} es vecino de s_j. Determinar entonces, el primer vecino que pertenece a otro pozo de atracción. Con esto, se podría ir generando pozos de atracción vecinos. Sin embargo, es un proceso computacionalmente muy costoso, por la cantidad de llamadas a búsqueda local que se requieren. Una variante es que en lugar de la noción de vecinos en pozos locales, se puede tomar una noción de vecinos más débiles. Otra idea es utilizar la noción de estados. Dado un mínimo local, se crea una perturbación que pertenezca al espacio de estados válidos y que genere un nuevo estado, en ese nuevo estado se aplica búsqueda local. Si el óptimo local encontrado pasa una prueba de aceptación (es mejor), entonces, se acepta el nuevo estado, si no se queda en el estado considerado actual.

La Búsqueda Local Iterativa no es reversible (no se permite regresar a los subespacios o espacios ya explorados); sin embargo, si se siguen perturbaciones determinísticas se pueden seguir ciclos cortos. Por lo mismo a las perturbaciones se les deben incluir aspectos aleatorios o ser adaptativas para evitar quedar ciclados. También en casos prácticos durante el criterio de aceptación de una solución, si después de un cierto tiempo de iteraciones en la búsqueda no se han logrado resultados, se puede empezar ILS desde una nueva solución.

Para aplicar ILS se dan cuatro componentes importantes que a continuación se describen, algunos de ellos pueden necesitar de afinación:

1. Una solución inicial s_o, que puede obtenerse de forma aleatoria o por medio de un algoritmo $greedy$ o glotón (que normalmente da mejores resultados)

2. Un procedimiento de *perturbación*, que modifica la solución actual s relacionándola con alguna solución intermedia s^i. La perturbación es el esquema que tiene ILS para escapar de mínimos locales. La fuerza de la perturbación está definida por el número de cambios realizados sobre la solución.

3. Un procedimiento de *búsqueda local* que regresa una solución mejorada s^m

4. un procedimiento del *criterio de aceptación* que decide a cual solución se le aplica la siguiente perturbación. Si no mantenemos una historia de lo ocurrido, el criterio de

aceptación simplemente considera la diferencia en costos entre s y s^m. El criterio de aceptación puede ser determinístico o se puede ajustar empíricamente conforme se soluciona el problema (por ejemplo, como se usa en recocido simulado). Además, el criterio de aceptación se usa para establecer un balance entre exploración (diversificación) y explotación (intensificación). La exploración se refiere a la búsqueda de soluciones en todo el espacio posible; la explotación se refiere a la búsqueda intensa de soluciones en una vecindad.

En el Algoritmo 4.1 se presenta la Búsqueda Local Iterativa

Algoritmo 4.1 Búsqueda Local Iterativa

1: **procedimiento** *Búsqueda Local Iterativa*
2: $s_o \leftarrow$ genera solución inicial ()
3: $s \leftarrow$ búsqueda local (s_o)
4: **Haga lo siguiente:**
5: $s^i \leftarrow$ perturbación(s,historia)
6: $s^m \leftarrow$ búsqueda local (s^i)
7: $s \leftarrow$ criterio de aceptación(s, s^m,historia)
8: **Repita hasta** alcanzar criterio de terminación

4.3. Algoritmos Genéticos

El término algoritmo genético se le atribuye a Holland y se le reconoce como el fundador, sin embargo, otros científicos trabajaron en ideas parecidas durantes los 60's.

En particular, Ingo Rechenberg y Hans-Paul Schwefel, en Alemania, desarrollaron las ideas de las estrategias evolutivas (Evolutions strategie), mientras que Bremermann y Fogel desarrollaron las ideas de programación evolutiva.

La parte común en estas estrategias es la mutación y la selección.

Los Algoritmos Genéticos (GA por sus siglas en Inglés de Genetic Algorithms), son conceptos que pueden tratarse como una familia de procedimientos de búsqueda adaptativos [Cruz-Chávez et al., 2010a]. A diferencia de las dos estrategias de tipo evolutivo, los algoritmos genéticos se concentran más en la combinación de soluciones para obtener mejores soluciones [Cruz-Chávez et al., 2010b].

Su nombre se deriva de los modelos de cambio genético en una población de individuos y su funcionamiento está basado en la evolución genética que presentan esos modelos. Esto es:

- Noción Darwiniana de aptitud (fitness), que influye en generaciones futuras
- Apareamiento, que producirá descendientes en generaciones futuras
- Operadores genéticos, que determinan la configuración genética de los descendientes (tomada de los padres).

Un punto clave de estos modelos, es que el proceso de adaptación no se hace cambiando incrementalmente una sola estructura, sino manteniendo una población de estructuras a partir de las cuales se generan nuevas estructuras usando los operadores genéticos.

Cada estructura en la población está asociada con una aptitud y los valores se usan en competencia para determinar qué estructuras serán usadas para formar nuevas estructuras.

Una de sus características es su habilidad de explotar información acumulada acerca de un espacio de búsqueda, inicialmente desconocido para guiar la búsqueda subsecuente a subespacios útiles [Cruz-Chávez et al., 2010b].

Su aplicación está enfocada sobre todo a espacios de búsqueda grandes, complejos y poco entendidos. El precio es que se puede necesitar un número grande de muestras para que se tenga suficiente información para guiar muestras subsecuentes a subespa-

cios útiles [Cruz-Chávez et al., 2006].

En su forma más simple, un GA está orientado al desempeño (i.e., hacer cambios estructurales para mejorar el desempeño).

Una de las ideas más importantes es definir estructuras admisibles en el sentido que estén bien definidas y puedan ser evaluadas.

Las diferencias con métodos tradicionales de búsqueda y optimización son:

- Trabajan con un conjunto de parámetros codificados y no con los parámetros mismos.
- Inician la búsqueda desde un conjunto de puntos, no de uno solo.
- Usan una función a optimizar en lugar de la derivada u otro conocimiento adicional.
- Usan reglas de transición probabilísticas no determinísticas.

La estructura de un GA esta compuesta principalmente por los siguientes módulos:

1. Módulo Evolutivo: mecanismo de decodificación (interpreta la información de un cromosoma) y función de evaluación (mide la calidad del cromosoma). Solo aquí existe información del dominio.
2. Módulo Poblacional: tiene una representación poblacional y técnicas para manipularla (técnica de representación, técnica de arranque, criterio de selección y de reemplazo). Aquí también se define el tamaño de la población y la condición de terminación.
3. Módulo Reproductivo: contiene los operadores genéticos.

En el Algoritmo 4.2 se muestra la estructura básica de un algoritmo genético tomado de [Rossi-Doria et al., 2003].

Algoritmo 4.2 Estructura básica de un Algoritmo Genético

```
1: Procedimiento GA
2: tiempo ← 0
3: Inicializa Población(tiempo)
4: Evalúa Población(tiempo)
5: Mientras no se alcanza condición de terminación hacer:
6:     tiempo ← tiempo + 1
7:     Construye Población(tiempo) a partir de Población(tiempo - 1) usando:
8:         • Selección
9:         • Modifica Población(tiempo) usando Operadores Genéticos
10:        • Evalúa Población(tiempo)
11:        • Reemplaza
```

Como se ha visto, los Algoritmos Genéticos o Algoritmos Evolutivos están basados en un modelo computacional del mecanismo de evolución natural [Baeck et al., 2000]. Los GA operan sobre una población de soluciones potenciales y comprenden tres grandes etapas: selección, reproducción y reemplazo. En la etapa de selección el individuo mejor dotado tiene mayor probabilidad de substituir que aquellos menos dotados o ser escogidos como padres para la siguiente generación, como en la selección natural. La reproducción se ejecuta como una recombinación y aplicación de operadores mutantes a los padres seleccionados: la recombinación toma partes de los dos miembros de la pareja para crear un nuevo individuo, mientras que la mutación normalmente hace pequeñas alteraciones en la copia de un solo individuo. Finalmente, los individuos de la población original se reemplazan por la creación de nuevos individuos, usualmente tratando de guardar los mejores individuos y desechando los peores. La explotación de buenas soluciones se garantiza por la etapa de selección, mientras que la explotación de nuevas zonas del espacio de búsqueda se efectúa en la etapa de reproducción, basados en el hecho de que la política de reemplazo permite la aceptación de nuevas soluciones que no necesariamente mejoran a las existentes.

Los algoritmos genéticos se han utilizado con éxito para resolver problemas de optimización combinatoria, incluyendo la programación de horarios de nivel superior. En el

estado del arte, en los algoritmos se usa con frecuencia información específica del problema para mejorar su desempeño, como por ejemplo mutación heurística [Ross et al., 1994] o alguna técnica constructiva guiada heurísticamente [Paechter et al., 1998].

Algunos puntos que se deben considerar al implementar algoritmos genéticos son:

- Codificación de los parámetros de un problema.
  Dentro de la codificación a veces se usan codificaciones que tengan la propiedad de que números consecutivos varíen a lo más en un bit (e.g., en el código Gray). En la codificación se busca idealmente que todos los puntos estén dentro del espacio de soluciones (sean válidos).

- Función de aptitud.
  Es la base para determinar que soluciones (poblaciones) tienen mayor o menor probabilidad de sobrevivir. Se tiene que tener un balance entre una función que haga diferencias muy grandes (y por lo tanto una convergencia prematura) y diferencias muy pequeñas (y por lo tanto un estancamiento).

- Criterios de tamaño de la población.
  Considerar el balance entre una población muy pequeña (y por lo tanto convergencia a un máximo local) y una población muy grande (y por lo tanto el requerimiento de muchos recursos computacionales). Los primeros intentos de tratar de estimar el tamaño de la población óptima basados en el teorema de esquemas resultaban en crecimiento exponenciales de la población con respecto al tamaño de gen. Experimentalmente se vió que esto no es necesario y se buscó el tamaño mínimo para poder alcanzar todos los puntos en el espacio de búsqueda.

  Para genes binarios, la probabilidad de que existe al menos un gen/individuo en cada punto es:

$$P_2 = (1 - (1/2)^{M-1})^l$$

  donde M es el tamaño de la población y l es el tamaño del gen. Esto sugiere que

con poblaciones de tamaño $O(logl)$ es suficiente.

Normalmente las poblaciones se seleccionan aleatoriamente, sin embargo, esto no garantiza una selección que nos cubra el espacio de búsqueda uniformemente.

Se pueden introducir mecanismos más sofisticados para garantizar diversidad en la población.

- Criterio de selección.
  Individuos son copiadas de acuerdo a su evaluación en la función objetivo (aptitud). Los más aptos tienen mayor probabilidad a contribuir con una o más copias en la siguiente generación (se simula la selección natural). Se pueden implementar de varias formas, sin embargo, la más común es la de simular con aleatoriedad, donde cada cadena tiene un espacio proporcional a su valor de aptitud.

$$Pr(h) = \frac{Aptitud(h)}{\sum_{j=1}^{N} Aptitud(h_j)}$$

  En lugar de seleccionar uno a la vez, se pueden generar N individuos separados uniformemente, los cuales se pueden generar por medio de un ciclo de programación. A esto se le conoce como “stochastic universal selection”.

  Otra alternativa es usar selección por torneo. Se seleccionan aleatoriamente N individuos que compiten entre sí seleccionando el mejor. En esta alternativa el mejor individuo siempre es seleccionado.

  Una ventaja de este esquema es que solo necesitamos comparar si un gen es mejor que otro, por lo que posiblemente no tenemos que evaluar la función de aptitud. Sin embargo, haciendo muestreo con reemplazo, existe una probabilidad de aproximadamente 0.368 ($\approx e^{-1}$) de que un gen no sea seleccionado.

  Finalmente, otro tipo de selección podría ser por categoría. Simplemente se ordenan

los genes. La probabilidad de seleccionar un gen categorizado como el k-ésimo ($P(k)$), en el caso de categorización lineal se tiene:

$$P(k) = \alpha + \beta k$$

donde α y β son constantes y

$$\sum_{k=1}^{M} (\alpha + \beta k) = 1$$

- Nueva población.
  Se pueden seleccionar individuos de la población actual, generar una nueva población y reemplazar con ella completamente a la población que se tenia (esquema generacional). También, a veces se mantienen los N mejores individuos de una población a la siguiente (esto parece ser la mejor opción). Si se conserva solo el mejor individuo se llama estrategia elitista, si se mantiene un subconjunto se habla de poblaciones traslapadas.

- Criterio de paro.

  Normalmente cuando un porcentaje alto de la población converge a un valor o después de un número fijo de evaluaciones en la función de aptitud. Si con ese valor no se llega a la medida esperada (si es que se conoce) o simplemente para tratar de mejorar la solución, entonces se toma una pequeña proporción y se inyecta "diversidad genética" (se generan aleatoriamente nuevos individuos), o inclusive se reemplaza completamente la población.

- Operadores genéticos.
  Normalmente se hace cruza seguida de mutación, pero se podría hacer una o la otra. Por ejemplo: algunos autores proponen hacer mucha cruza al principio e incrementar la mutación conforme pasa el tiempo. La decisión se puede considerar dependiendo de la experiencia que se encuentre en la literatura. Los operadores se pueden utilizar o crear según las posibilidades y estrategias del diseñador.

4.4. Recocido Simulado

El Recocido Simulado (SA por sus siglas en Inglés de Simulated Annealing) es una técnica sencilla de búsqueda local probabilística, tiene una buena reputación y lo han utilizado diferentes grupos de investigación para encontrar soluciones en problemas de optimización. Su nombre viene del hecho que simula el enfriamiento [Kirkpatrick et al., 1983, Cerny, 1985] de una colección de átomos vibrando en caliente -los átomos a mayor temperatura, mayor vibración-. El SA como proceso físico se aplica a materiales sólidos en el cual:

- Se incrementa la temperatura a un material sólido hasta cambiar su estado original a líquido.
- Se decrementa la temperatura muy lentamente hasta alcanzar un estado base de mínima energía. Con esto, se busca obtener una estructura cristalina (sólida) lo más regular posible.

Normalmente se aplica a metales que inicialmente son calentados a temperaturas muy altas y luego se dejan enfriar muy lentamente para que sus moléculas se acomoden en una estructura regular (cristalina). De forma sencilla a este proceso se le llama *templado*.

La metaheurística de recocido simulado se puede aprovechar en diferentes áreas del conocimiento, debido a sus características de funcionamiento. Kirkpatrick y sus colegas [Kirkpatrick et al., 1983] la introdujeron hace más de veinticinco años en la Optimización Combinatoria como un esquema de mejoramiento iterativo [Aarts and Korst, 1989, Dowsland and Adenso Díaz, 2001].

El Recocido Simulado como método de optimización está basado en el método de Monte Carlo [Laarhoven and Aarts, 1992] y es útil para simular estocásticamente el enfriamiento lento de un material que pasa del estado líquido al sólido. Este proceso de simulación, se aprovecha también en la optimización combinatoria para obtener soluciones cercanas al óptimo con montos razonables de tiempo de cómputo, con la ventaja de que estas soluciones no dependen de la configuración inicial.

El núcleo del algoritmo de recocido simulado lo constituye el algoritmo de Metrópolis [Metrópolis et al., 1953, Dowsland and Adenso Díaz, 2001] para realizar la simulación del enfriamiento físico en la transición de líquido a sólido, la estructura de Metrópolis incluye formulación de la probabilidad de aceptación de Boltzmann de la mecánica clásica. Por cierto, en 1953, Metrópolis y sus colegas publicaron el primer artículo describiendo el uso del método de Montecarlo en simulación de procesos termodinámicos.

En el Algoritmo 4.3 se muestra la estructura del algoritmo de Metrópolis, donde k_B indica la constante de Boltzmann y T la temperatura.

Algoritmo 4.3 Metrópolis

1: Dado un estado i con energía E_i,
2: genera un nuevo estado j mediante un mecanismo de perturbación // (perturbación: pequeña distorsión del estado i).
3: calcula energía E_j del nuevo estado j.
4: **Si** $(E_j - E_i) \leq 0$ **entonces:**
5: $i \leftarrow j$ (acepta el estado nuevo)
6: **En otro caso:**
7: **Si** $e^{\left(-\frac{E_j - E_i}{k_B \cdot T}\right)} > random[0, 1)$ **entonces:**
8: $i \leftarrow j$

Lo que hace el algoritmo de Metrópolis es: a partir de un estado de energía (energía de referencia) actual generar un nuevo estado (perturbación), muy cercano al actual, y calcular su energía, aceptará ese nuevo estado si tiene menor o igual energía; pero si la energía es mayor, lo aceptará con cierta probabilidad, todo lo anterior dependiendo de la temperatura T. Si se realiza este proceso durante muchas transiciones se puede llegar a lo que se conoce en termodinámica como un equilibrio térmico, el cual está caracterizado por la distribución de boltzmann.

Durante el proceso del SA se ejecuta varias veces el algoritmo de Metrópolis, por eso al SA se le considera como una secuencia de ciclos de Metrópolis. En el Algoritmo 4.5 se muestra la estructura del algoritmo de Recocido Simulado.

Algoritmo 4.4 El Recocido Simulado

1: obtener una configuración inicial i y una temperatura inicial T
2: **Mientras** no se satisfaga el *criterio de paro* **hacer:**
3: **Mientras** no se satisfaga el *criterio de lazo interior* **hacer:**
4: ejecutar algoritmo *Metrópolis* y guardar configuración en i.
5: reducir temperatura T.
6: regresa mejor configuración (i) encontrada.

A continuación se presenta la teoría de la distribución de Boltzmann y del criterio de aceptación de estados de energía, para este fin y por objetividad en la escritura de ecuaciones, en aquellas ecuaciones que estén elevadas a alguna potencia con la base e, esta base se substituirá por el término exp, por ejemplo: $e^{\left(-\frac{E_j-E_i}{k_B \cdot T}\right)}$ se escribirá como $exp\left(-\frac{E_j-E_i}{k_B \cdot T}\right)$.

La distribución de Boltzmann da la probabilidad P de que el sólido esté en el estado i con energía E_i a temperatura T y está dada por:

$$P_T\{X = i\} = \frac{1}{Z(T)} exp\left(-\frac{E_i}{k_B T}\right) \tag{4.1}$$

donde X es la variable estocástica que indica el estado actual del sólido, T es la temperatura del sistema, k_B es la constante de Boltzmann. El factor $exp\left(-\frac{E_i}{k_B \cdot T}\right)$ se conoce como el factor de boltzmann. A medida que la temperatura disminuye, la distribución de Boltzmann se concentra en los estados con menor energía y, por último, cuando la Temperatura se aproxima a cero, sólo los estados de mínima energía tienen una probabilidad no nula de ocurrencia. $Z(T)$ es un factor de normalización, que se conoce como la función de partición, que depende de T y k_B. La función de partición se define como:

$$Z(T) = \sum_i exp\left(-\frac{E_i}{k_B T}\right) \tag{4.2}$$

donde la sumatoria recorre todos los estados macroscópicos posibles.

Se puede mostrar que haciendo una reducción muy pequeña de la temperatura T y generando suficientes transiciones en cada parámetro de temperatura se puede alcanzar el equilibrio térmico o sea la configuración óptima en optimización combinatoria.

Dado un estado i, la probabilidad de aceptación de un estado nuevo j se expresa como sigue:

$$P_{c_k}\{\text{acepta } j\} = \begin{cases} 1 & \text{si } f(j) \leq f(i) \\ exp\left(-\dfrac{f(j) - f(i)}{c_k}\right) & \text{si } f(j) > f(i) \end{cases}$$

donde $c_k \in \mathbb{R}^+$ indica el parámetro de control. $c_k = k_B T$ donde k_B es la constante de Boltzmann y T la temperatura.

Una transición consiste en:

1. Aplicación del mecanismo de generación, y
2. Aplicación del mecanismo de aceptación.

Inicialmente con valores grandes de c_k la probabilidad de aceptación de estados nuevos es muy alta y va disminuyendo esta conforme disminuye el valor de c_k, esta variación de valor es debida a la disminución de la temperatura T. Al tender c_k a cero se dejan de aceptar estados.

En el recocido simulado, la velocidad de convergencia de la probabilidad de aceptación de estados de energía esta determinada por la *longitud de la cadena de Markov* L_k y la *reducción de la temperatura* de enfriamiento T, representados por el criterio de lazo interior (ciclos) y por la reducción aplicada a T respectivamente.

La búsqueda local en optimización combinatoria es el equivalente del proceso de enfriamiento muy lento, que también se conoce en la práctica como el templado con amortiguamiento.

Un sistema físico de muchas partículas puede verse como un “ensamble” estadístico.

Si el ensamble es estacionario, el cual se logra al establecer el equilibrio térmico, su densidad es función de la energía del sistema y la probabilidad de estar en un estado i con energía E_i está dado por la distribución de Boltzmann.

Dada una instancia de un problema combinatorio y una estructura de vecindad adecuada, después de un número de transiciones suficientemente largo para un valor fijo de c_k, aplicando el criterio de aceptación, el algoritmo de recocido simulado encuentra una solución $i \in S$ con probabilidad igual a:

$$P_{c_k}\{X = i\} \equiv q_i(c_k) = \frac{1}{N_0(c_k)} exp\left(-\frac{f(i)}{c_k}\right)$$

donde X es una variable estocástica que indica la solución actual y

$$N_0(c_k) = \sum_{j \in S} exp\left(-\frac{f(j)}{c_k}\right)$$

indica una constante de normalización.

Dada una instancia de un problema de optimización combinatoria, una estructura de vecindad adecuada y una distribución estacionaria equivalente a la distribución de Boltzmann, entonces:

$$\lim_{c_k \to 0} q_i(c_k) \equiv q_i^* = \frac{1}{\mid \mathcal{S}_{opt} \mid} \mathcal{X}_{\mathcal{S}_{opt}}(i)$$

donde $\mathcal{S}_{opt}$ indica el conjunto de soluciones óptimas globales, y $\mathcal{X}_{\mathcal{S}_{opt}}(i)$ es la función característica definida como:

$$\mathcal{X}_{\mathcal{S}_{opt}}(i) = \begin{cases} 1 & \text{si } i \in \mathcal{S}_{opt} \\ 0 & \text{de otra forma} \end{cases}$$

El resultado es importante porque garantiza una convergencia asintótica hacia el conjunto de soluciones óptimas del algoritmo de recocido simulado bajo la condición de que la distribución estacionaria se obtenga para cada valor de c_k.

La probabilidad de encontrar la solución óptima se incrementa al decrementar continuamente c_k. La acción de búsqueda de la solución óptima relacionado con el decremento

continuo representa una convergencia asintótica. A continuación se trata la relación de la Convergencia Asintótica con las cadenas de Markov para el algoritmo de Recocido Simulado.

El algoritmo de recocido simulado se puede modelar matemáticamente usando Cadenas de Markov. Una Cadena de Markov es una secuencia de sucesos, donde la probabilidad del resultado de un suceso depende sólo del resultados del suceso anterior. En la teoría relacionada con las cadenas de Markov se utiliza más el término “evento” en lugar de “suceso”; sin embargo aquí se utiliza “suceso” para evitar confusión, ya que el término “evento” se utiliza en las variables para el UCTP.

Sea $X(k)$ una variable estocástica que denota el resultado del k-ésimo suceso. Entonces la transición de probabilidad en el k-ésimo suceso para cada par i, j de resultados se define como:

$$P_{ij}(k) = \mathbf{P}\{X(k) = j \mid X(k-1) = i\}$$

La matriz $P(x)$ cuyos elementos están dados por la fórmula anterior se le llama: matriz de transición.

Sea $b_i(k)$ la probabilidad del resultado i en el k-ésimo suceso, o sea: $b_i(k) = \mathbf{P}\{X(k) = i\}$. Entonces, $b_i(k)$ se define como:

$$b_i(k) = \sum_l b_l(k-1)P_{li}(k)$$

A continuación se dan algunas propiedades de las Cadenas de Markov. Se dice que una Cadena de Markov es:

- Finita, si tiene un conjunto finito de resultados.
- No homogénea, si las probabilidades de transición dependen del número del suceso k.
- Homogénea, si las probabilidades de transición son independientes del número del suceso.

- Predecible, si para cada par de soluciones $i, j \in S$ existe una probabilidad positiva de alcanzar j a partir de i en un número finito de sucesos.

En recocido simulado, la consecuencia de un suceso corresponde a una transición y los posibles resultados corresponden al conjunto de posibles soluciones.

Las probabilidades de transición del algoritmo de recocido simulado están definidas como:

$$\forall i, j \in S : P_{ij}(k) = P_{ij}(c_k) = \begin{cases} G_{ij}(c_k)B_{ij}(c_k) & \text{si } i \neq j \\ 1 - \sum_{l \in S, l \neq i} P_{il}(c_k) & \text{si } i = j \end{cases}$$

donde:

$G_{ij}(c_k)$ indica la probabilidad de generar una solución j a partir de una solución i.

$$\forall i, j \in S : G_{ij}(c_k) = G_{ij} = \frac{1}{\Theta}\mathcal{X}_{S_i}(j)$$

donde $\Theta = \mid S_i \mid$, para toda $i \in S$.

$B_{ij}(c_k)$ es la probabilidad de aceptar la solución j una vez que fue generada a partir de la solución i.

$$\forall i, j \in S : B_{ij}(c_k) = exp\left(-\frac{(f(j) - f(i))^+}{c_k}\right)$$

donde, $\forall b \in \mathbb{R}, b^+ = b$ si $b > 0$ y $b^+ = 0$ de otra forma.

Lo que se quiere probar es que, bajo ciertas condiciones, el algoritmo de recocido simulado converge asintóticamente al conjunto de soluciones óptimas, i.e.:

$$\lim_{k \to \infty} P\{X(k) \in S_{opt}\} = 1$$

Una parte esencial para la prueba de convergencia es la existencia de una distribución estacionaria única. Tal distribución existe sólo si se cumplen ciertas condiciones en

la cadena de Markov asociada al algoritmo.

Una distribución estacionaria de una cadena de Markov finita con matriz de transición P se define como un vector q cuyo i-ésimo componente está dado por:

$$q_i = \lim_{k\to\infty} P\{X(k) = i \mid X(0) = j\}, \text{ para toda } j$$

Si existe esa distribución estacionaria, entonces,

$$\begin{aligned}\lim_{k\to\infty} b_i(k) &= \lim_{k\to\infty} P(X(k) = i) \\ &= \lim_{k\to\infty} \sum_j P(X(k) = i \mid X(0) = j)P(X(0) = j) \\ &= q_i \sum_j P(X(0) = j) = q_i\end{aligned}$$

Se tenía que:

$$\lim_{c_k\to 0} q_i(c_k) \equiv q_i^* = \frac{1}{\mid \mathcal{S}_{opt} \mid} \mathcal{X}_{\mathcal{S}_{opt}}(i)$$

Entonces, como:

$$\lim_{c_k\to 0} \lim_{k\to\infty} P(X(k) = i) = \lim_{c_k\to 0} q_i(c_k) = q_i^*$$

Por lo tanto:

$$\lim_{c_k\to 0} \lim_{k\to\infty} P(X(k) \in S_{opt}) = 1$$

Aunque el desarrollo matemático antes expuesto, no necesariamente es una prueba específica, si muestra las ideas principales que se pueden considerar en la prueba de convergencia.

Todas las pruebas usan la condición de reversibilidad o también llamada de balance detallado. Lo que dice es que si tenemos una matriz de transición P asociada a una cadena de Markov finita, predecible, aperiódica y homogénea. Entonces una distribución es estacionaria para la cadena de Markov si se satisface que:

$$q_i P_{ij} = q_j P_{ji}, \text{ para toda } i, j \in S$$

Las pruebas de convergencia nos dicen que el algoritmo de recocido simulado requiere un número infinito de transiciones para aproximarse lo suficiente a una distribución estacionaria en cada temperatura T. Hay que recordar que esta temperatura T esta inmersa en el parámetro de control de temperatura $c_k = k_B T$ donde k_B es la constante de Boltzmann. Esto involucraría la generación de secuencias infinitas para valores descendentes de la temperatura T en el parámetro de control de temperatura c_k -una secuencia infinita para cada valor de T-. Esta generación infinita es completamente imposible de poner en práctica. Sin embargo, se puede formular el algoritmo de recocido simulado como una secuencia de cadenas de Markov de longitud *finita* que *convergan* al *conjunto de soluciones óptimas* si el enfriamiento se hace lo suficientemente lento.

El proceso se puede describir mediante la combinación de cadenas de Markov homogéneas en una sola cadena de Markov no homogénea. Dicho de otra forma, *la secuencia de cadenas de Markov homogéneas infinitamente largas se reducen a una sola cadena de Markov no homogénea infinita*.

Como en la práctica no se puede garantizar llegar a la solución óptima, se hacen aproximaciones con un número de transiciones finitas y una cantidad finita de decrementos de temperatura que arrojan soluciones subóptimas. También, para el parámetro de control de temperatura c_k se utiliza solo la temperatura T ya que como se vió $c_k = k_B T$, donde k_B es constante y para la simulación en optimización combinatoria se considera unitaria, por lo tanto $c_k = T$.

Además de lo anterior expuesto, para que Recocido Simulado funcione bien, se requiere definir un *mecanismo de enfriamiento* que especifique:

- Una secuencia finita de valores para controlar la temperatura T, a saber:
    1. Un valor inicial,
    2. Una función de decremento, y
    3. Un valor final (condición de paro).
- Un número finito de transiciones de L_k (longitud finita de la cadena de Markov) para

cada valor de la temperatura T. Este número de transiciones corresponde con el número de ciclos de Metrópolis.

Una idea clave en las aproximaciones es llegar a un cuasi equilibrio. Esto es, tratar que la distribución de probabilidad de las soluciones, después de un número finito de transiciones, sea "suficientemente cercana" al equilibrio térmico o distribución estacionaria. Para que esto ocurra, se busca hacer un balance entre la longitud de las cadenas de Markov y los decrementos realizados en el parámetro de control de temperatura. Ya que, un decremento elevado de T requerirá un considerable número de transiciones para restablecer el cuasi equilibrio y viceversa.

A continuación se da la *analogía* de los principales términos del recocido simulado que se utilizan en termodinámica física y su correspondencia en optimización combinatoria:

- La *partición* más probable para un estado macroscópico en termodinámica equivale a la *distribución* más probable de las soluciones en optimización combinatoria. Entonces, cuando se habla de alcanzar el *equilibrio térmico*, corresponde a encontrar la *distribución estacionaria* más probable. Es decir, la solución más cercana a la óptima, sin excluir que se pueda obtener la óptima.

- Los diferentes estados de energía E en termodinámica equivale al conjunto de soluciones S en optimización combinatoria. Entonces, un *estado* i corresponde con una *solución* i.

- La *energía* E_i de un estado i corresponde con la *función de costo* $f(i)$ de una solución i.

- El parámetro de control de temperatura c_k para controlar el proceso de enfriamiento en termodinámica, corresponde con la variable T para controlar el descenso de la temperatura después de un número preestablecido de ciclos de Metrópolis (número de transiciones finito o longitud de L_k).

Kirkpatrick propuso un mecanismo de *enfriamiento* que considera los parámetros de control, los cuales requieren de un *análisis de sensibilidad* que se puede realizar por medio del procesamiento experimental para obtener la sintonía de los parámetros del

recocido simulado. A continuación se muestra el mecanismo de enfriamiento propuesto por Kirkpatrick:

- Valor inicial del parámetro de control de la temperatura. Empezar con un entero positivo pequeño y continuar multiplicándolo por un factor mayor a 1 hasta que las transiciones generadas (ciclos) sean casi todas aceptadas.

- Decremento del parámetro de control de temperatura c_k (o T). Entonces, $c_{k+1} = \propto \cdot c_k$, donde $\propto$ es una constante cercana a 1 (rango: 0.8 - 0.99).

- Longitud de las cadenas de Markov. Hacer una longitud L_k fija (de otra forma cuando $c_k \to 0$, $L_k \to \infty$).

- Valor final del parámetro de control de temperatura. Terminar cuando la solución obtenida no mejora después de un número determinado consecutivo de cadenas.

Aparte de la propuesta de Kirkpatrick, en la literatura se pueden encontrar algunas otras proposiciones de esquemas de enfriamiento, aunque muchos autores no divulgan claramente sus procedimientos.

En general, para la implementación del algoritmo de recocido simulado se requiere especificar tres componentes:

1. Hacer la representación del problema:

   a) Representar el espacio de la solución

   b) Expresar la función de costo que represente adecuadamente el costo (valor) de las soluciones

2. Realizar el mecanismo de transición:

   a) Generar una nueva solución (cercana a la actual)

   b) Calcular la diferencia de costo (lo más común es calcular la diferencia en costo contra la solución anterior)

c) Tomar la decisión de aceptar o no una solución de acuerdo a la probabilidad de aceptación P ya explicada

$$P(\text{ aceptación }) = \begin{cases} 1 & \text{si } \triangle f \leq 0 \\ exp\left(-\frac{\triangle f}{T}\right) & \text{si } \triangle f > 0 \end{cases}$$

3. Establecer el mecanismo de enfriamiento y dar los parámetros de las variables siguientes para la sintonía del recocido simulado:

   a) El *valor inicial* de la temperatura T

   b) *Función de decremento* de la temperatura T

   c) La *longitud* L_k de las cadenas de Markov (número de ciclos de Metrópolis)

   d) El criterio de *paro*. Temperatura T final

En el Algoritmo 4.5 se muestra un ejemplo del algoritmo de recocido simulado que utiliza el grupo Metaheuristic Network [Rossi-Doria et al., 2003] para procesar la búsqueda de soluciones de instancias problema del UCTP. El algoritmo está compuesto de dos partes (o fases) y en cada fase usan un Recocido Simulado. En la primera fase tratan exclusivamente con las restricciones duras (hcv) y donde Th representa la temperatura en la región infactible, en la segunda fase tratan sólo con las restricciones suaves (scv) y donde Ts representa la temperatura en la región factible. Durante la *Fase de restricciones duras*, el algoritmo busca una solución (factible o no). Si encuentra la solución factible, pasa a la *Fase de restricciones suaves* donde trata de optimizar la solución.

Algunas de las características importantes que contiene el recocido simulado y que hay que considerar, son:

- El algoritmo es sencillo y fácil de implementar
- Es aplicable a una gran cantidad de problemas
- Aunque el algoritmo es sencillo su adaptación para los problemas no es siempre trivial y a veces hay que hacer reformulaciones
- Su eficiencia depende mucho del esfuerzo que se ponga en la representación de la estructura de vecindad y del mecanismo de enfriamiento que se incluya

Algoritmo 4.5 Ejemplo de algoritmo de Recocido Simulado

entrada: Una instancia problema I
$i \leftarrow$ solución inicial aleatoria
{Fase de restricciones duras}
$Th \leftarrow Tho$ // Tho: es temperatura inicial en fase de restricciones duras
Mientras el tiempo límite no se alcance y $hcv > 0$ **hacer:**
    Actualiza la temperatura
    $j \leftarrow$ Genera una solución vecina de i
    Si $f(j) < f(i)$ **entonces:**
        $i \leftarrow j$.
    En otro caso:
        $i \leftarrow j$ con probabilidad $P(T, i, j) = e^{-\left(\frac{f(j)-f(i)}{T}\right)}$.
    $i_{mejor} \leftarrow$ la mejor entre i e i_{mejor}.
{Fase de restricciones suaves}
$Ts \leftarrow Tso$ // Tso: es temperatura inicial en fase de restricciones suaves
Mientras el tiempo límite no se alcanza y $scv > 0$ **hacer:**
    Actualiza la temperatura
    $j \leftarrow$ Genera una solución vecina i
    Si $hcv = 0$ en j **entonces:**
        Si $f(j) < f(i)$ **entonces:**
            $i \leftarrow j$.
        En otro caso:
            $i \leftarrow j$ con probabilidad $P(T, i, j) = e^{-\left(\frac{f(j)-f(i)}{T}\right)}$.
        $i_{mejor} \leftarrow$ la mejor entre i e i_{mejor}.
salida: Una solución optimizada i_{mejor} para I

4.5. Búsqueda Tabú

La metaheurística de Búsqueda Tabú [Glover and Laguna, 1998] es una estrategia para resolver problemas de optimización combinatoria. La metodología que se incluyó en Búsqueda Tabú, como en otras más metodologías de búsqueda, utilizan como base en su estructura la Búsqueda Local (LS por sus siglas en Inglés de Local Search), y que los implementadores aplican con algunas variantes. En general, la metodología de Búsqueda Local, vista como un algoritmo, funciona de la siguiente manera:

1. Seleccionar una $x \in X$ inicial
2. Seleccionar algún $s \in N(x)$ (vecino)
3. Si $c(s) < c(x)$ entonces, $x \leftarrow s$ y regresa al punto (2).
   Sino, x es un óptimo local y termina el proceso.

Búsqueda Tabú combina búsqueda local con una heurística para evitar parar en mínimos locales y evitar quedar atrapados en ciclos. La idea básica de Búsqueda Tabú es continuar con LS cuando se llega a un mínimo local al permitir movimientos que no necesariamente mejoran la solución.

Otra característica importante que tiene Búsqueda Tabú, es que para evitar regresar a soluciones pasadas (repeticiones) y ciclarse, usa una memoria temporal, llamada *lista tabú*, que guarda la historia reciente de la búsqueda.

En resumen, Búsqueda Tabú tiene dos elementos clave que combina para su funcionamiento, estos son:

- *Restricciones Tabú*. Que se consideran en la búsqueda local para restringir la búsqueda al clasificar ciertos movimientos como prohibidos (tabú), para evitar caer en soluciones recientemente generadas, y

- *Criterio de aspiración* (aspiration criteria). Mecanismo de memoria a corto plazo que se usa como alternativa para liberar la búsqueda (como almacén de olvido estratégico)

Para el proceso en Búsqueda Tabú, se crea un subconjunto $T \subseteq S$ con la información histórica y se extiende t iteraciones en el pasado.

El acceso a la información de T puede ser por medio de condiciones a cumplir (i.e., no necesariamente un índice de soluciones pasadas).

En Algoritmo 4.6 se muestran las instrucciones que forman un algoritmo básico de Búsqueda Tabú.

Algoritmo 4.6 Instrucciones de un algoritmo básico de Búsqueda Tabú

1: **Selecciona** un estado $x \in X$ inicial y sea $x^* \leftarrow x$, $k \leftarrow 0$ (contador de iteración) y $T \leftarrow 0$.
2: **Si** $S(x) - T = 0$ **entonces:**
3: Ve a 9.
4: **En otro caso:**
5: $k \leftarrow (k+1)$ y selecciona $s_k \in S(x) - T$ tal que $s_k(x) = OPTIMO(s(x) : s \in S(x) - T)$.
6: $x \leftarrow s_k(x)$.
7: **Si** $c(x) < c(x^*)$ **entonces:**
8: $x^* \leftarrow x$. // Donde x^* es la mejor solución encontrada hasta el momento
9: **Si** se agotó el numero de iteraciones $\| S(x) - T = 0$ **entonces:**
10: Para el proceso.
11: **En otro caso:**
12: Actualiza T // Añade el movimiento actual a la lista tabú y posiblemente elimina el elemento más viejo
13: Regresa a 2.

Como se podrá ver, el algoritmo de búsqueda tabú cuenta con las siguientes características:

- Las soluciones dependen de como se actualiza T
- No hay condición de óptimo local

- Se busca la “mejor” solución en cada paso (función ÓPTIMO), en lugar de alguna opción que mejore la solución

ÓPTIMO puede ser:

$$c(s_k(x)) = \text{ mínimo}(c(s(x)) : s \in S(x) - T)$$

La función ÓPTIMO da la mejor solución o la menos peor sujeta a la lista tabú.

En principio, se podría tomar alguna opción que mejore la solución (en caso de que sea difícil encontrar la mejor), pero normalmente se sigue la estrategia más agresiva. La razón principal es que los óptimos locales no presentan problemas.

Normalmente la lista tabú se implementa como una lista circular, añadiendo elementos en la posición 1 y eliminando los que sobrepasen la posición t (para una t fija).

Una forma efectiva de implementar la lista T sería:

$$T = \{s^{-1} : s = s_h \text{ para } h > k - t\}$$

donde k es el número de iteración y s^{-1} es el movimiento inverso de s.

Por lo que el paso de actualización de T sería: $T := T - s_{k-t}^{-1} + s_k^{-1}$.

En general, lo que se trata de evitar es caer en estados de solución previos. Esto no quiere decir que se tenga que escoger una t muy grande.

En la práctica T no toma la forma anterior: (i) s^{-1} previene un conjunto más grande de movimientos, (ii) por consideraciones de memoria es deseable guardar sólo un subconjunto de atributos que caracterizan el movimiento.

Bajo estas condiciones, la lista tabú representa una colección de movimientos C_h.

Cuando la lista $S - T = \emptyset$, se pueden eliminar los elementos más viejos de T para permitir continuar con el proceso.

Para el control de los Niveles de Aspiración, con la lista tabú se evitan ciclos si se proyecta el camino para ir de x a $s(x)$ si: (1) $s(x)$ ya ha sido visitada antes (repetir caminos es muy caro en memoria e implementación); (2) el movimiento s ya se aplicó a x antes; (3) el movimiento s^{-1} ya se aplicó a $s(x)$ antes.

La lista tabú puede prohibir movimientos deseables que no produzcan ciclarse o también nos puede llevar a un punto en donde no es posible moverse.

Todos los algoritmos de búsqueda tabú permiten revocar o cancelar tabúes.

A estos se les llama criterios de aspiración (aspiration criteria), que permiten movimientos, aunque sean tabú.

Lo más común es permitir movimientos que producen una mejor solución que la mejor solución actual, osea si se hace un movimiento que va de una x a $s(x)$ si $c(s(x)) < MEJOR(c(x))$.

Si se piensa en términos de atributos para describir estados, se pueden tener listas tabú para cada atributo y una función de aspiración que depende de cada atributo.

Si uno o más atributos pasan la prueba individual de aspiración, entonces se puede asumir que los demás atributos automáticamente también la pasan.

La idea de intensificación es buscar más porciones del espacio que aparentan ser mejores o más prometedoras.

Cada determinado tiempo se puede realizar un proceso de intensificación.

Normalmente se reinicia la búsqueda a partir de la mejor solución actual. Algunas opciones son:

- Mantener fijos los componentes o atributos que parecen mejores
- Cambiar el esquema de vecindad

Una extensión es considerar los comportamientos de los patrones producidos por una lista tabú.

Uno de los efectos de ésto es analizar por ejemplo movimientos oscilatorios evaluando lo atrayente de los movimientos dependiendo de la localización y dirección de la búsqueda.

Con ésto se puede por ejemplo especificar un número de movimientos necesarios en una cierta dirección antes de permitir algún retorno.

Las funciones de memoria a mediano plazo sirven para registrar y comparar atributos de las mejores soluciones obtenidas durante un período de búsqueda.

Los atributos comunes puedes servir para guiar nuevas soluciones a espacios en donde existan tales atributos.

Otra consideración respecto a la exploración es la diversificación. Las funciones de memoria a largo plazo tratan de diversificar la búsqueda, empleando principios más o menos contrarios a los de memoria a mediano plazo.

La idea es explorar regiones que contrastan fuertemente con las regiones exploradas hasta el momento.

No se trata de inyectar diversidad aleatoria, sino tomando en cuenta el proceso de búsqueda realizado hasta ese momento.

Para escapar de atracciones fuertes se requiere de un esfuerzo adicional. Algunas posibilidades para lograrlo son:

1. Imponer restricciones más fuertes en las condiciones tabú para excluir un número más grande de movimientos
2. Usar información de cuándo el proceso apunta hacia arriba y tratar de favorecer movimientos que apunten en esa dirección
3. Penalizar movimientos que usan atributos muy usados en el pasado

Una vez que se sale de latracciónor, se pueden “relajar” las condiciones.

Conceptualmente, lo que tratan de estimar es una distancia de escape del óptimo local.

Se pueden incorporar también elementos probabilísticos, para preferir movimientos con cierta probabilidad.

Algunos resultados muestran que el tamaño de la lista tabú entre 5 y 12 (y alrededor de 7) es adecuado. Sin embargo, para algunos problemas se tiene que determinar cuál es la mejor.

Existen variaciones que se pueden aplicar de Búsqueda Tabú como puede ser por ejemplo la Búsqueda Tabú Reactiva (o RTS por sus siglas en Inglés de Reactive Tabu Search). La RTS adapta el tamaño de la lista a las propiedades del problema de optimización.

Se almacenan las configuraciones visitadas y sus números de iteración, por lo que se puede calcular el número de repeticiones de una configuración y el intervalo entre dos visitas.

El mecanismo de reacción básico aumenta la lista tabú cuando se repiten configuraciones y lo reduce cuando no se necesita aumentar.

Se añade un mecanismo de diversificación de memoria a largo plazo cuando se sospecha de una atracción fuerte.

Para entender las atracciones o también llamados atractores, los mínimos locales se pueden ver como atracciones en la dinámica de la política de máxima pendiente.

Por otro lado, también se pueden tener ciclos, que se repiten continuamente.

Una tercera posibilidad es que la trayectoria esté restringida a una cierta área del espacio de búsqueda (atracciones caóticas).

Existe en sistemas dinámicos el concepto del exponente de Lyapunov. Si tenemos una función g que mapea un punto en el paso n a un punto en el paso $n+1$, $g^k(x)$ se define como el mapeo obtenido al interactuar g, k veces.

Si se empieza con configuraciones cercanas x_0 y $x_0 + \epsilon$, el exponente de Lyapunov λ se define por la relación:

$$\epsilon e^{n\lambda} \approx || \; g^n(x_0 + \epsilon) - g^n(x_o) \; ||$$

Si $\lambda > 0$, los puntos iniciales divergen exponencialmente, y si las trayectorias se mantienen dentro de una región en el espacio, se obtiene lo que se llama caos determinístico.

La trayectoria parece aleatoria pero el sistema es determinístico. En tal caso, aunque no se tienen ciclos, sólo se visita una región pequeña del espacio.

El evitar ciclos no debe de ser la única meta, también se debe continuar con la búsqueda de soluciones de mejor calidad.

Una forma de tratar de mejorar la calidad de las soluciones, podría ser como lo muestran las instrucciones del siguiente algoritmo:

1. Evaluar todos los posibles movimientos elementales a partir de la configuración actual

2. Buscar la última configuración y decidir si se hace un mecanismo de escape

3. Si no hay escape, realizar el mejor movimiento

4. Si hay escape, el sistema deberá entrar en un ciclo de movimientos aleatorios cuya duración dependa del promedio de los movimientos en ciclos detectados

Cuando se hace una repetición, el mecanismo básico de reacción aumenta el tamaño de la lista tabú. Después de un número suficiente de repeticiones se elimina cualquier tipo de ciclo.

Esto no es suficiente para evitar trampas caóticas. Para ésto, se tiene otro mecanismo más lento que cuenta el número de configuraciones que son repetidas. Cuando el número es mayor a una cierta constante se entra al mecanismo de diversificación.

Por otro lado, también se tiene un proceso lento que reduce la lista tabú si el número de iteraciones es más grande que el promedio de movimientos entre ciclos desde el último cambio.

Cuando la lista tabú es tan grande que todos los movimientos son tabú y ninguno satisface el criterio de aspiración, entonces se reduce la lista tabú.

La estrategia de escape se basa en realizar un conjunto de movimientos aleatorios proporcionales al número promedio de movimientos entre ciclos.

Para evitar regresar a la región, todos los movimientos aleatorios se vuelven tabú.

En experimentos realizados, si se elimina el mecanismo de escape, el sistema frecuentemente queda atrapado y no encuentra la solución óptima.

Se puede concluir que la Búsqueda Tabú es una metaheurística de búsqueda local que se basa en estructuras de memoria especiales para evitar el entrampamiento en mínimos locales y lograr un balance efectivo entre la explotación y la exploración durante la búsqueda. También, TS tiene gran poder para encontrar soluciones de buena calidad de problemas difíciles de optimización combinatoria, problemas que se encuentran en gran variedad de aplicaciones. Más precisamente y considerando la Búsqueda Tabú para la solución de problemas como el UCTP, el TS permite la búsqueda para explorar soluciones que no degraden el valor de la función objetivo; pero solo en aquellos casos donde estas soluciones no sean olvidadas durante el proceso. Esto último se puede lograr, si se guardan en memoria las últimas soluciones, es decir, si se conservan las acciones que transformaron una solución en la siguiente. El nombre de ”tabú”se le da, ya que una de las características propias del TS es que, cuando se ejecuta un movimiento *el retroceso se considera tabú* para las siguientes iteraciones l, donde l es el tamaño de la lista tabú L. Entonces, una solución es prohibitiva si se obtiene por la aplicación de un movimiento tabú a la solución actual.

En el Algoritmo 4.7 se muestra un ejemplo de algoritmo clásico de Búsqueda Tabú adaptado para obtener soluciones del UCTP. En el algoritmo, un movimiento es olvidado si al menos uno de los eventos involucrados se ha movido antes menos de l pasos. El status de la longitud l de la lista L (lista Tabú) se establece según el número de eventos dividido por una constante conveniente k. Esto, con el propósito de disminuir la probabilidad de generación de ciclos y de reducir el tamaño de la vecindad para una exploración rápida, se considera un conjunto variable de vecinos (en este ejemplo se considera el conjunto del 10 % de la vecindad). Más aun, para explorar el espacio de búsqueda en forma más eficiente, a la Búsqueda Tabú, se le incluye usualmente un criterio de aspiración para aceptar un movimiento aun si éste ha sido marcado como tabú. Se ejecuta un movimiento tabú si el movimiento mejora la mejor solución conocida actualmente.

Algoritmo 4.7 Ejemplo de Búsqueda Tabú para el UCTP

1: Ingresar una instancia problema I
2: $s \leftarrow$ una solución inicial aleatoria
3: $L \leftarrow 0$, donde L indica la lista tabú
4: **Mientras** el tiempo límite no se alcance **hacer:**
5: **Para** $i = 0$ a 10 % de la vecindad **hacer:**
6: $s_i \leftarrow s$ después del i-esavo movimiento
7: Calcula correctividad de $f(s_i)$
8: **Si** $\exists s_j | f(s_j) < f(s)$ y $f(s_j) \leq f(s_i) \forall i$ **entonces:**
9: $s \leftarrow s_i$
10: $L \leftarrow L \cup E_i$ // Donde E_i es el conjunto de eventos movidos para obtener la solución s_i
11: **En otro caso:**
12: $s \leftarrow$ el mejor movimiento no tabú entre todas las s_i
13: $L \leftarrow L \cup E_b$ // Donde E_b es el conjunto de eventos movido por el mejor movimiento no tabú
14: $s_{best} \leftarrow$ la mejor solución hasta ahora
15: Salida: Una solución optimizada s_{best} para I

Capítulo 5

Propuesta de Solución para el UCTP

La propuesta de solución que aquí se presenta, es la creación e implementación del algoritmo GSTT y es una de las principales aportaciones en esta investigación para obtener soluciones satisfactorias del Problema de Programación de Cursos en una Universidad (UCTP). La creación de este algoritmo se realizó considerando que fuese competitivo con aquellos que se encuentran actualmente en la literatura. Además, el obtener soluciones satisfactorias, implica que computacionalmente las soluciones sean factibles, es decir, que cumplan lo mejor posibles las necesidades, las restricciones y los requerimientos de los elementos que intervienen en la problemática. Por lo tanto, la finalidad de obtener soluciones satisfactorias del UCTP es obtener los horarios escolares o tablas de horarios para una universidad o escuela superior que *satisfagan a la mayoría de las personas* que intervienen en la realización de los eventos y que los recursos, que normalmente son limitados, se distribuyan de la mejor manera posible.

A continuación se presenta la implementación del Algoritmo GSTT, se inicia con la Representación del Modelo que sirve para construir su estructura, la metodología que se aplicó para encontrar soluciones factibles y la Estrategia de Asignación de Recursos, que es también una de las aportaciones de este trabajo doctoral, la cual fue ideada para tratar de obtener mejores soluciones factibles. Se presenta también, el Diseño del Algoritmo con la estructura que se puso en práctica en la implementación. Además se incluye, el algoritmo de Recocido Simulado con la Estructura de Vecindad que se incluyó para realizar la perturbación necesaria al moverse en el espacio de búsqueda para tratar de

encontrar las mejores soluciones posibles, esta estructura de vecindad es una más de las aportaciones de este trabajo. Por último, se incluye el Análisis de Sensibilidad para obtener los parámetros de sintonia.

5.1. Algoritmo GSTT

Como ya se mencionó en el Capítulo 2, el problema de la programación de cursos a nivel universitario o de escuelas superiores implica la asignación de recursos escolares como pueden ser: el conjunto de eventos (clases, seminarios, exámenes, tareas, tutoriales, etc.) y recursos como salones, laboratorios, estudiantes, profesores, máquinas, computadoras, etc., dentro de un número limitado de periodos de tiempo (horarios, sesiones), tal que se satisfaga un objetivo y un conjunto de restricciones.

La problemática de asignar los recursos debido a las características de los componentes humanos y materiales, a sus características, a las facilidades que pueden proporcionar y a las necesidades para funcionar, así como las restricciones que se deben cumplir, hacen difícil de obtener soluciones aceptables en tiempos de cálculo computacional que se puedan considerar aceptables. Entonces, el objetivo de obtener un algoritmo que pueda obtener soluciones aceptables en tiempos que se puedan considerar competitivos con los existentes en la teoría, requiere de un buen análisis, una buena estrategia para la programación de los recursos, que además no sea estructuralmente muy compleja para que el tiempo de cálculo computacional no sea muy elevado. También, que la perturbación o estrategia de búsqueda local sea lo más sencilla posible para que no exploten los tiempos de búsqueda de soluciones durante la optimización de una solución. Con todo esto en mente, hemos creado el algoritmo GSTT y que a continuación presentamos como una propuesta de solución para el problema de la Programación de Cursos en una Universidad.

5.1.1. Representación del modelo

El algoritmo GSTT esta compuesto de varios módulos, los cuales representan las actividades principales que ejecuta el algoritmo para tratar de obtener soluciones de las

Clases	pequeña	mediana	grande
Número de eventos	100	400	400
Número de salones	5	10	10
Número de características	5	5	10
Características aproximadas por salón	3	3	5
Porcentaje de características usadas	70	80	90
Número de estudiantes	80	200	400
Número máximo de eventos semanal por estudiante	20	20	20
Número máximo de estudiantes por evento	20	50	100

Cuadro 5.1: Parámetros de las 3 clases de instancias

instancia problema que se den como dato de entrada.

El GSTT acepta como datos de entrada, cualquiera de las instancias problema que se conocen en la literatura como benchmarks, debido a que son utilizadas como paradigma por la comunidad científica. Estas instancias están disponibles en la página del grupo Metaheuristic Network [Rossi-Doria et al., 2003]. Este grupo creó un generador de instancias para producir las instancias problema con diferentes características y valores diferentes de parámetros dependiendo de los recursos a programar. El grupo garantiza que todas las instancias que producen tienen una solución óptima, es decir, una solución sin violación de restricciones duras o suaves. El generador accede a una semilla aleatoria y además toma ocho parámetros que se dan desde la línea de comandos, estos parámetros especifican los recursos de la instancia. Si se da al generador la misma estructura de parámetros y la misma semilla, producirá la misma instancia problema; pero si se utiliza la misma estructura de parámetros con una semilla diferente entonces producirá una instancia diferente.

El grupo Metaheuristic Network, basados en sus investigaciones y pruebas sobre instancias problema, hicieron una clasificación de estas instancias problema o benchmarks que reunieron en tres clases de acuerdo al grado de dificultad y tamaño según el número de recursos a programar y las llamaron como: pequeñas, medianas y grandes. La lista con las tres clases de instancias se pueden ver en el Cuadro 5.1.

Estas tres clases de instancias se utilizan como datos de entrada para el GSTT. El uti-

lizar estas instancias como paradigmas permitirá que los resultados que se obtengan del algoritmo GSTT se puedan comparar con aquellos resultados que existen en la literatura y poder hacer una evaluación del desempeño del GSTT.

En la teoría, la mayoría de investigadores representan su modelo para resolver el UCTP como una estructura bidimensional, como la que se mostró en la representación simbólica de la Sección 5.1.1 en donde se distribuyen los recursos materiales y humanos. La representación bidimensional, esta compuesta por un eje de periodos de tiempo u horarios y el otro eje por lo días. La tabla es una estructura que considera las actividades académicas semanales donde se programan los recursos escolares como son las clases, conferencias, seminarios, presentaciones, exámenes, Etc. que en general se les llama eventos, también considera, los salones donde se llevan a cabo los eventos, la distribución de estudiantes que toman las clases según sus niveles y necesidades y todos aquellos recursos que se planifican en una tabla de horarios organizados en el tiempo. Por lo tanto, la tabla de horarios es un resumen del producto que se obtiene de la organización, distribución y asignación de los recursos, similar a la tabla mostrada en el Cuadro 3.1, con una estructura de 45 timeslots (9x5 de periodos de tiempo por días)que en términos computacionales le llamamos: obtener una solución.

En la estructura bidimensional, antes de hacer la asignación de eventos en los timeslots, se realiza un preproceso para designar a cada evento los posibles salones donde se podrá efectuar ese evento considerando las características y tamaño de los salones debido a las necesidades de cada evento. Posteriormente, para realizar la programación de eventos utilizan un algoritmo de emparejamiento que se encarga de asignar los eventos en los timeslots y relaciona cada evento con un salón a partir de la lista preprocesada de los posibles salones antes designados. Posteriormente, si hubiera aun eventos no programados, los van tomando en orden secuencial y asignan cada uno dentro de un salón que cumpla las características requeridas y el aforo adecuado. Todo lo anterior según lo explicado en la Sección 5.1.1. Esta técnica de asignación de recursos, principalmente de timeslots (horarios) y días, no considera ninguna restricción dura o suave que este relacionada con los estudiantes, lo que da como resultado que se tenga una probabilidad

bastante alta desde el inicio de tener una solución infactible. Por lo tanto, la búsqueda de una solución factible implica reacomodar gran cantidad de eventos con múltiples movimientos para cada timeslot y entre timeslots, ya que con esta estructura bidimensional se asignan varios eventos en un timeslot, lo que da como consecuencia la toma de tiempos elevados de cálculo computacional. Hay que pensar que este tipo de movimientos de sacar o introducir varios elementos en un timeslot, se manejaría con una estructura de datos de tipo pila.

A diferencia del arreglo bidimensional para representar la estructura de las soluciones del UCTP que se encuentran en la teoría, la representación del GSTT se concibe como una estructura tridimensional. Los ejes de periodos y días son similares a los de la teoría; sin embargo el GSTT considera un tercer eje formado por los salones. Así, la estructura tridimensional se asemeja a varias rebanadas en paralelo -una tabla rectangular por salón- (como el pan en rebanadas) para formar la tercera dimensión. Por lo tanto, en el GSTT se tendrá una tabla de horarios por cada salón y los salones formarán la tercera dimensión. Este tipo de estructura tridimensional tiene ciertas ventajas contra la estructura bidimensional, ya que se pueden asignar los eventos de manera atómica, es decir, se puede colocar solo un evento por timeslot, lo que facilita la distribución de los eventos en el espacio tridimensional y para hacer búsquedas o reubicaciones. Computacionalmente, la diferencia de manejar un arreglo bidimensional a un arreglo tridimensional no tiene gran impacto de tiempo computacional durante el recorrido de los arreglos (a fin de cuentas, son arreglos más grandes los tridimensionales; pero unidimensionales ambos), sin embargo, para la búsqueda o reubicación de eventos en la estructura tridimensional considerando solo un evento por timeslot, el movimiento de quitar e introducir un solo elemento en cada timeslot resulta menos complicado ya que con la estructura tridimensional no habría necesidad de utilizar estructuras de datos tipo pila. En lo que corresponde a la búsqueda de tener una solución factible o de convertir una solución factible a infactible, que implica verificar el cumplimiento de restricciones duras, la estructura tridimensional se considera resulta de la misma complejidad que una bidimensional, ya que se tiene que verificar el cumplimiento de restricciones para cada evento/estudiante y en este caso depende de la cantidad de eventos y estudiantes que contenga la instancia problema. Por

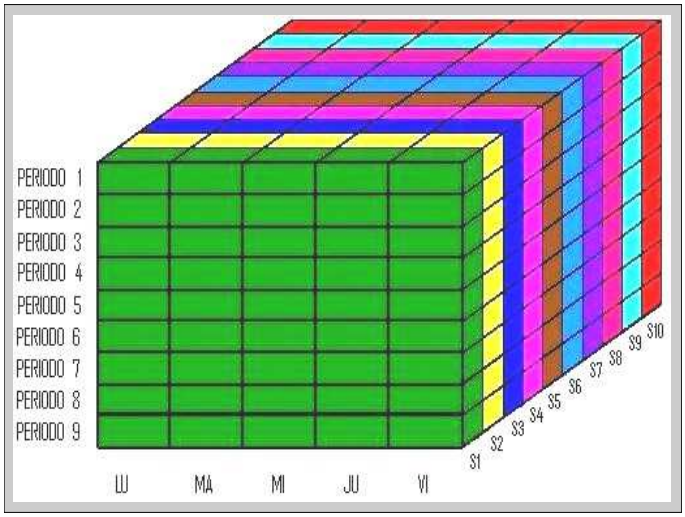


Figura 5.1: Estructura tridimensional del GSTT

lo tanto, es independiente de la estructura bidimensional o tridimensional.

La estructura tridimensional se puede ver en la Figura 5.1, donde se presenta un ejemplo de una abstracción que tiene: 9 periodos de tiempo (PERIODO 1 a PERIODO 9) , 5 días de la semana (LU, MA, MI, JU, VI) y 10 salones (S1 a S10), con lo que se tendría un espacio de 450 (9x5x10) timeslots disponibles. Recuérdese que una instancia grande considera máximo 400 eventos para asignar, entonces se tendría espacio suficiente para colocar un evento por timeslot.

5.1.2. Estrategia de asignación de recursos

La forma tradicional de abordar el problema de Asignación de Recursos y de encontrar soluciones se hace de la siguiente manera:

Primero, se identifican los recursos que se van programar. Segundo, la asignación de recursos se hace de forma aleatoria o de forma semialeatoria-dirigida, distribuyéndolos en todo el espacio de posibilidades, o también, utilizando una estrategia particular de asignación. En ambos casos de asignación, la solución que se obtenga tendrá un porcentaje muy alto de ser infactible y remotamente puede ser factible desde el inicio. Posteriormente, se aplica una metaheurística y una perturbación para tratar de encontrar soluciones

factibles y mejorarlas, proceso que normalmente consume bastante tiempo, debido a que en estos casos es común alojar todos los eventos sin considerar inicialmente algunas o ninguna de las restricciones duras. La perturbación se aplica para hacer la búsqueda de soluciones en una vecindad o en todo el espacio de posibles soluciones. Entonces, la metaheurística modera la estrategia de la perturbación considerando el tiempo de computación y el espacio de búsqueda.

Para desarrollar la metaheurística del GSTT se trató el problema de programar los recursos de manera diferente a la forma tradicional utilizada en la teoría. La Estrategia de Asignación de Recursos que se puso en práctica en el diseño del algoritmo GSTT para programar los horarios se estructuró de acuerdo al siguiente criterio:

> *Con el fin de obtener y conservar la factibilidad de una solución, la asignación de un horario para cada evento se efectuará sólo si, al seleccionar un timeslot, se cumplen las cinco restricciones duras (H1 a H5).*

Para cumplir con las restricciones duras, se tomó en cuenta el Modelo Matemático que se desarrollo en el Capítulo 3 para la programación de los eventos.

La implementación de la Estrategia de Asignación de Recursos se trata a continuación (Capítulo 5.1.3) con el diseño del algoritmo GSTT y la descripción de su funcionamiento.

5.1.3. Diseño del algoritmo

Un algoritmo es una secuencia lógica finita de pasos necesarios para ejecutar una tarea específica tal como la solución de un problema. Entonces, un algoritmo representa la solución de un problema. A continuación se muestra el diseño del algoritmo GSTT con su estructura, formada por los diferentes módulos que identifican sus actividades, así como la secuencia de ejecución de estas actividades para tratar de obtener soluciones de las instancias problema que se den como dato de entrada.

El GSTT acepta como datos de entrada, cualquiera de las instancias problema que se

conocen en la literatura como benchmarks [Rossi-Doria et al., 2003]. Estas instancias se clasifican de acuerdo con su tamaño como: chicas, medianas o grandes (Ver Cuadro 5.1). En el Capítulo 5.1.1 se dió una explicación sobre estas instancias problema.

Las soluciones que se obtengan con el algoritmo GSTT corresponderán a las tablas de horarios. Entonces, de acuerdo con la Estrategia de Asignación de Recursos, que se mencionó en el Capítulo 5.1.2, del conjunto de soluciones obtenidas, se escogerá la mejor solución y se le aplicará el proceso de optimización con la metaheurística de Recocido Simulado. A continuación se da la explicación de las actividades que constituyen el GSTT, el cual se muestra en el Algoritmo 5.1 y su diagrama de flujo en la Figura 5.2. La explicación hace referencia a la secuencia de los números de instrucciones que componen el algoritmo.

1. *Seleccionar la instancia de entrada* I. Selecciona la instancia del problema que se desea resolver. La instancia se incluye por medio de un menú para indicar el nombre del archivo donde se encuentran los datos de entrada.

2. *Leer la instancia* I. Se leen los datos desde el archivo seleccionado para transferirlos a instancia I.

3. ***Para*** *(numsol = 0; numsol* $<$ *MaxSol; numsol++)* ***hacer:*** Esta instrucción, que forma parte de una estructura de programación ”for”, verifica la condición en cada ciclo de ejecución para procesar el número de soluciones (MaxSol) que se hayan solicitado. En caso de que la condición sea verdadera, se realizan 16 posibles actividades (instrucciones de la 4 a la 19), en caso contrario se continúa con la instrucción 20. Este conjunto de instrucciones -de la 3 a la 19- es la parte más importante del algoritmo en cuanto al proceso de creación de soluciones de la instancia seleccionada como problema.

4. *Inicializar a ceros los arreglos TT y ExA*. Con esta instrucción se inicializan los arreglos TT y ExA y demás estructuras de datos que se emplean para manipular y mover los datos durante la creación, mejora y optimización de una solución como son las estructuras ENA1 (Eventos No Asignados uno) y ENA2 (Eventos No Asignados dos). TT y ExA se declaran como arreglos tridimensionales debido a la organización

Algoritmo 5.1 GSTT

1: Seleccionar la instancia de entrada I
2: Leer la instancia // Instancia del problema a resolver
3: **Para** (numsol = 0; numsol $<$ MaxSol; numsol++) **hacer:**
4: Inicializar a ceros los arreglos TT y ExA
5: ExA $\Leftarrow$ Instancia I
6: *Asigna* al azar eventos de ExA en timeslots vacíos de TT que cumplan restricciones duras
7: *Reemplaza* al azar algunos eventos de TT por algunos de ENA1 que cumplan restricciones duras
8: *Reasigna* eventos de ENA2 en timeslots vacíos de TT, verificando que cumplan restricciones duras // Produce una solución parcial factible
9: Eventos remanentes en ENA2 se asignan en timeslots vacíos de TT sin verificar restricciones. // La solución podría volverse infactible
10: **Para** (intentos=0; (intentos$<$intentosMax) || (solución no es factible); intentos++) // Busca rehacer factible la solución, **hacer:**
11: Seleccionar al azar de TT un par de timeslots ocupados con eventos
12: Permutar el par de eventos si cumplen restricciones duras.
13: **Si** La solución es factible **entonces:**
14: **Para** (intentos = 0; intentos $<$ intentosMax; intentos++) // Reacomodar eventos, **hacer:**
15: Seleccionar al azar de TT un par de timeslots ocupados con eventos
16: Permutar el par de eventos si cumplen restricciones duras y se mejora la solución.
17: Guardar solución en BancoSoluciones
18: Imprimir: Se encontró una solución *factible*.
19: **En otro caso:**
20: Imprimir: Se encontró una solución *infactible*.
21: **Si** hay soluciones en BancoSoluciones **entonces:**
22: Elegir la mejor solución i obtenida
23: *Optimizar* i con Recocido Simulado
24: Imprimir la Solución Optimizada i.
25: **En otro caso:**
26: Imprimir: El GSTT no encontró solución factible.

de datos que se guardan en ellos. En TT se guardan los eventos programados y que servirán para obtener las Tablas de Horarios -una tabla por salón-. EnrecocidoSimuladoParaGSTT ExA se guarda la memoria de los eventos que resten por asignar. El arreglo ENA1 se utiliza para guardar los eventos que no se puedan asignar en TT en el primer intento; el arreglo ENA2 se utiliza para guardar los eventos que no se puedan asignar en TT en un segundo intento.

5. *ExA* $\Leftarrow$ *Instancia* I. Los datos de entrada que se encuentran en la instancia problema I se copian al arreglo ExA.

6. *Asigna al azar eventos de ExA en timeslots vacíos de TT que cumplan restricciones duras*. Por medio de una selección aleatoria de los eventos y de los timeslots, se intenta programar la mayor cantidad posible de eventos por asignar (ExA) en timeslots vacíos del arreglo TT, siempre y cuando se cumplan las restricciones duras; aquellos eventos que no se puedan asignar en un número máximo (Max) de intentos se guardarán en el arreglo ENA1.

7. *Reemplaza al azar algunos eventos de TT por algunos de ENA1 que cumplan restricciones duras*. También por selección aleatoria de timeslots, se procede a sustituir algunos eventos en timeslots ocupados de TT por eventos de ENA1, la sustitución se hará siempre y cuando se cumplan las restricciones duras; aquellos eventos de ENA1 que no se puedan reemplazar en un número máximo (Max) de intentos se guardarán en el arreglo ENA2.

8. *Reasigna eventos de ENA2 en timeslots vacíos de TT, verificando que cumplan restricciones duras*. La ejecución de esta instrucción reasigna en el arreglo TT los eventos que aun resta programar y que se guardaron en ENA2 durante el proceso anterior con la instrucción Reemplaza (7). Para el proceso, se hace una selección FIFO de los eventos del arreglo ENA2 y se reasignan por selección aleatoria en timeslots vacíos de TT. Esta reasignación de cada evento se hará siempre y cuando el timeslot seleccionado de TT cumpla las restricciones duras. La búsqueda de un timeslot adecuado para reasignar cada evento se repite en un número máximo (Max) de intentos; si se alcanza el número máximo de intentos sin poder reasignar el evento, entonces se deja éste en su lugar original del arreglo ENA2 y se procede

a tratar de programar el siguiente evento.

Con estas tres últimas instrucciones (6, 7 y 8) se pone en práctica la “Estrategia de Asignación de Recursos” que se explicó en el Capítulo 5.1.2.

Al concluir esta actividad, se habrán asignado en TT la mayor cantidad de eventos del conjunto de eventos a programar y, debido a que la asignación se realizó de acuerdo a la Estrategia de Asignación de Recursos, entonces se tendrá en el arreglo TT una *solución parcial factible*.

9. *Eventos remanentes en ENA2 se asignan en timeslots vacíos de TT sin verificar restricciones*. Los eventos que aun no se han podido programar y que se encuentran en ENA2 se asignan en timeslots vacíos de TT sin considerar las restricciones duras. Debido a esta forma de asignar los horarios, *la solución podría volverse infactible* -lo más seguro-. Se recurre a esta última actividad, cuando se presenta el problema de no poder encontrar un horario adecuado en la programación de algunos eventos, a pesar de haber ejecutado tres intentos intensivos durante la ejecución de las instrucciones 6, 7 y 8.

10. ***Para*** *(intentos = 0; (intentos* $<$ *intentosMax)* $||$ *(solución no es factible); intentos++)* ***hacer:*** Con esta instrucción se trata de convertir la solución, si es infactible, en factible por medio de un reacomodo de los eventos del arreglo TT que aun no cumplan las restricciones duras. La instrucción, que forma parte de una estructura de programación “for”, verifica la condición en cada ciclo de ejecución para procesar hasta un número máximo de intentos (intentosMax) o si la solución deja de ser infactible, lo que ocurra primero. En caso de que la condición sea verdadera se realizan las instrucciones 11 y 12, en caso contrario se continúa con la instrucción 13.

11. *Seleccionar al azar de TT un par de timeslots ocupados con eventos*. Se selecciona aleatoriamente un par de timeslots de TT que estén ocupados con eventos.

12. *Permutar el par de eventos si cumplen restricciones duras*. Esta instrucción permuta el par de eventos alojados en los timeslots seleccionados en 11. Este proceso se

realiza siempre y cuando al permutar se cumplen las restricciones duras.

13. ***Si*** *la solución es factible* ***entonces:*** Esta instrucción tiene una estructura de programación “if ... then ...”, en su proceso verifica si a este nivel se alcanzó obtener una solución factible -que se tendrá almacenada en TT-. En caso positivo, tratará de hacer un reacomodo de eventos con un límite determinado de intentos -el objetivo será: mejorar la solución-, para esto se ejecutarán las instrucciones 14, 15 y 16. Posteriormente, ejecutará las instrucciones 17 y 18 para guardar la solución e informar que encontró una solución factible; en caso negativo, saltará para ejecutar la instrucción 19 donde imprimirá que no encontró una solución factible.

14. ***Para*** *(intentos = 0; intentos* $<$ *intentosMax; intentos++)* ***hacer:*** Esta instrucción “for” trata de hacer un reacomodo de eventos con un límite de ciclos determinado por “intentosMax”. El objetivo es de mejorar la solución que se encuentra en la estructura TT. Mientras se cumpla la condición propuesta, se realizarán las instrucciones 15 y 16.

15. *Seleccionar al azar de TT un par de timeslots ocupados con eventos*. La selección se realiza siempre y cuando los timeslots de TT estén ocupados con eventos. Esta instrucción forma parte de la instrucción 14.

16. *Permutar el par de eventos si cumplen restricciones duras y se mejora la solución*. Esta instrucción permuta el par de eventos alojados en los timeslots seleccionados en 15. Este proceso se realiza siempre y cuando al permutar se cumplen las restricciones duras y se mejora la solución; en caso contrario se dejan los eventos en su lugar original. Esta instrucciones también forma parte de la instrucción 14.

17. *Guardar solución en BancoSoluciones*. La solución obtenida se guarda en el Banco de Soluciones de donde posteriormente se escogerá la mejor de las soluciones guardadas (obtenidas).

18. *Imprimir: Se encontró una solución factible*. Se envía el mensaje para avisar que se encontró una solución factible. El número de soluciones factibles que se obtengan dependerá del número solicitado por menú; pero en el Banco de Soluciones únicamente se guardarán las que sean factibles.

19. *Imprimir: Se encontró una solución infactible*. Se llega a esta instrucción en caso de no cumplirse la condición de la instrucción 13. Entonces, se da aviso de haber encontrado una solución que no es factible.

20. ***Si** hay soluciones en BancoSoluciones **entonces:*** A esta instrucción se llega después de haber concluido la instrucción "*for*" (**para**) de la instrucción 3, donde se verifica buscar y encontrar soluciones factibles según el número solicitado (MaxSol) por menú. La instrucción es una estructura de programación "if ... then ..." que verifica si se cumple la condición de existencia de soluciones factibles en el Banco de Soluciones. En caso de haber soluciones, se ejecutan las instrucciones 21, 22 y 23; en caso contrario se salta al proceso 24 para imprimir que no se encontró solución factible.

21. *Elegir la mejor solución i obtenida*. Con esta instrucción se elige la mejor solución -factible- almacenada en el Banco de Soluciones. La mejor solución corresponderá a aquella que tenga el menor valor de Función Objetivo del conjunto de soluciones factibles obtenidas (soluciones óptimas).

22. *Optimizar i con Recocido Simulado*. A la solución i se le aplica el proceso de optimización con la metaheurística de Recocido Simulado que tiene integrada como perturbación, la Estructura de Vecindad propuesta para la búsqueda local. El algoritmo de búsqueda local propuesto se muestra en el Cuadro 5.2.

23. *Imprimir la Solución Optimizada i*. Esta instrucción imprime la solución optimizada i y que corresponde a las tablas de horarios -producto del proceso del algoritmo GSTT-.

24. *Imprimir: El GSTT no encontró solución factible*. Esta instrucción se ejecutará sólo si no encontró soluciones almacenadas en BancoSoluciones y quiere decir que el algoritmo GSTT no logró encontrar alguna solución factible.

Como se podrá ver en el algoritmo GSTT, del número de soluciones solicitadas, si obtuvo alguna(s), el GSTT escoge la mejor solución para aplicarle el proceso de optimización con la metaheurística de Recocido Simulado. En el Capítulo 5.2, trataremos más sobre el Recocido Simulado y la Estructura de Vecindad que se integraron en el GSTT

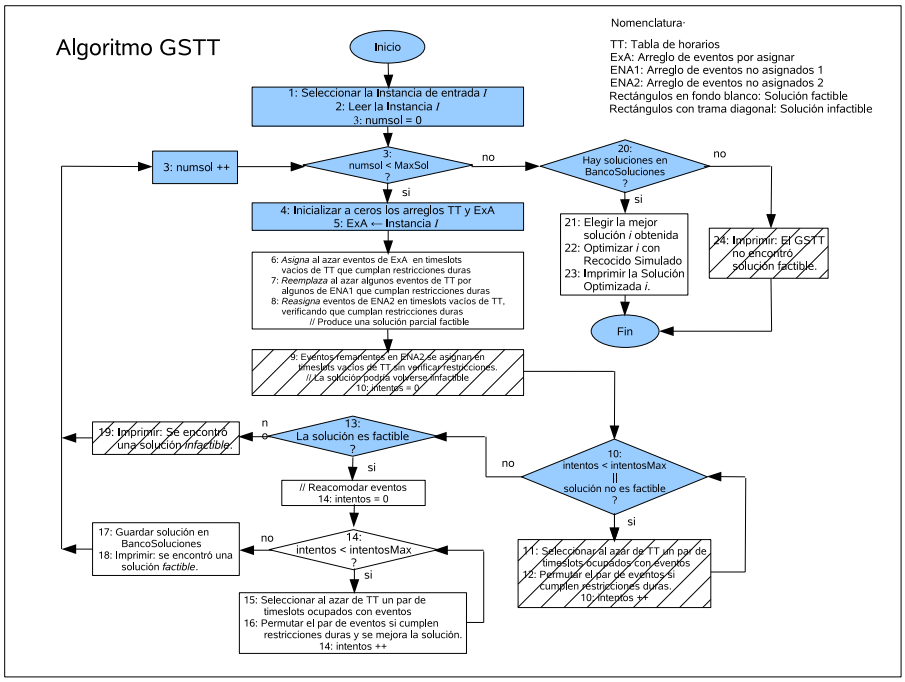


Figura 5.2: Diagrama de flujo del algoritmo GSTT

para optimizar las soluciones.

Para aumentar la descripción del algoritmo GSTT y visualizar el flujo lógico de sus actividades, en la Figura 5.2 se incluye su diagrama de flujo. Para facilitar el seguimiento de las instrucciones del algoritmo y su relación con las actividades como diagrama de flujo, a ambos descriptores se les dió la misma numeración.

Como ya se mencionó antes, con las instrucciones (o actividades) marcadas con los números 6 7 y 8 del Algoritmo 5.1 se puso en práctica la primera aportación de esta investigación. A esta aportación se le dió el nombre de *Estrategia de Asignación de Recursos* y de la cual se hizo su presentación en el Capítulo 5.1.2. En la Figura 5.3 se presenta un diagrama de flujo para dar más detalle de las tres actividades.

Al inicio del diseño del algoritmo GSTT, se tuvo la necesidad de incluir una actividad que se nombró *Crea Instancia* [Cruz-Rosales et al., 2010] y que se tenía en el lugar de la instrucción actual llamada *Seleccionar la instancia de entrada* I. La implementación de

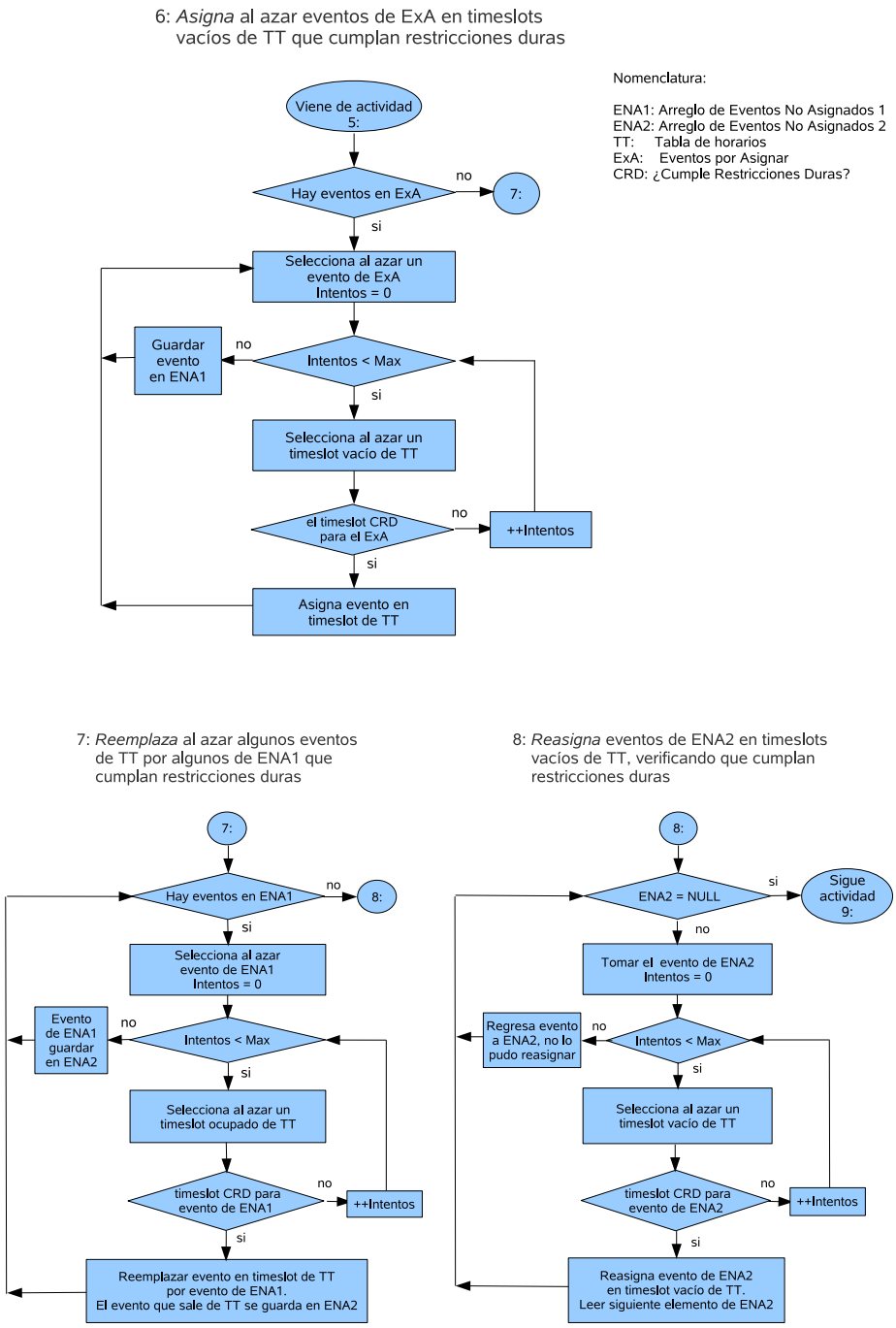


Figura 5.3: Detalle de las actividades 6, 7 y 8 del algoritmo GSTT y que producen una solución parcial factible

.

Crea Instancia se hizo debido a la necesidad de realizar pruebas parciales de funcionamiento al algoritmo GSTT durante las primeras etapas de diseño e implementación. Crea instancia se encargaba de crear la instancia de entrada del tamaño que se le solicitaba por medio de un menú. De esta forma, dependiendo del rango de las variables que se proporcionaban para cada lista de recursos, se generaba el tamaño de la instancia deseada (pequeña, mediana o grande). Para obtener diferente instancia, *cada vez se hacía una generación aleatoria de números enteros*. De esta forma se obtenía la cantidad de: eventos, salones, estudiantes y facilidades que formaban los conjuntos de recursos de una instancia. El conjunto de recursos que forman el problema característico del UCTP, se pueden ver en detalle en la formulación que se presenta en el Capítulo 3.1 del Modelo Matemático del UCTP. Desde luego, en el modelo también se consideraban: los días y periodos de tiempo que forman los timeslots y que se incluyen como parámetros fijos dentro del código del programa.

La actividad Crea Instancia se descartó, cuando se integró en el algoritmo GSTT la metaheurística de Recocido Simulado y la perturbación propuesta para la búsqueda de mejores soluciones y así obtener soluciones optimizadas. Para este propósito, en lugar de la actividad Crea Instancia se utilizan las instancias problema conocidas en la teoría como benchmarks. Estos benchmarks se pueden obtener en [http://www.metaheuristic.org/, 2009] o en [http://iridia.ulb.ac.be/Supp/IridiaSupp2002-001/index.html, 2009], ambas direcciones del grupo Metaheuristic Network.

5.2. Recocido Simulado para el GSTT

La metaheurística de Recocido Simulado es una variante de la búsqueda local que permite movimientos ascendentes para evitar quedar atrapado prematuramente en un óptimo local. Si se examina el SA, se verá que mas que un algoritmo, es una estrategia heurística con una perturbación de búsqueda local que necesita de varias decisiones para que quede totalmente diseñado. Estas decisiones son cuatro variables que tienen una gran influencia en la calidad de las soluciones y se deberán modular por medio de un análisis de sensibilidad para fijar los parámetros de sintonía.

Se escogió la metaheurística de Recocido Simulado para tratar de optimizar las soluciones que se obtienen con el algoritmo GSTT debido a las características de funcionamiento para obtener soluciones cercanas al óptimo con una cantidad de tiempo de cómputo razonable [Cruz-Chávez, 2004, Cruz-Chávez et al., 2006] y que estas soluciones no dependen de la configuración inicial.

La estructura de recocido simulado en optimización combinatoria se basa en fases e iteraciones. El número de fases, lo determina el usuario. Cada fase, se compone de un conjunto de iteraciones continuas L_k (número de transiciones que representa la longitud de la cadena de Markov), durante las cuales, la temperatura se mantiene constante. Al final de cada fase, la temperatura es decrementada. El número de fases se termina hasta que se cumpla el criterio de parada que se controla con la temperatura final, que también la determina el usuario.

La teoría más amplia del SA se dió en el Capítulo 4.4. Aquí y a continuación, se presenta el algoritmo de recocido simulado que se implementó para optimizar las soluciones que genera el algoritmo GSTT.

- Sea i una *solución inicial* y que dentro del algoritmo de SA se convierte en *solución de referencia* para comparar las soluciones cercanas que se encuentren durante el proceso de optimización.
- Sea $N(i)$ el entorno o *vecindad* de una solución i, donde N es el conjunto de soluciones cercanas o vecinas a i, entonces $f(j)$ será el *costo* de una solución j cercana a i.
- Sea To la *temperatura inicial* o parámetro de control, se utiliza para mantener la temperatura durante cada fase del proceso de optimización. Cada fase está compuesta de un número de ciclos de Metrópolis en analogía a la longitud de la cadena de Markov L_k.
- Sea $\propto$ una *función de reducción de la temperatura* o coeficiente de control. Esta función se utiliza para reducir la temperatura To al término de cada fase de búsqueda

de soluciones, en analogía a la reducción de la temperatura y latencia para alcanzar el equilibrio térmico durante el enfriamiento lento.

- Sea ncM el número máximo de *iteraciones* que representa la longitud de la cadena de Markov L_k o número de ciclos de Metrópolis. Se utiliza ncM como parámetro de paro para completar una fase. En analogía al criterio de aproximarse cada vez más (en cada fase) a una solución óptima.

- Sea Tf la *temperatura final* o criterio de paro de SA. En analogía a obtener una solución óptima o aproximada a la óptima.

Los parámetros de sintonía de las variables To, $\propto$, ncM y Tf se obtuvieron por el proceso de *análisis de sensibilidad* que se verá en el Capítulo 5.2.2 y en el Algoritmo 5.2 se muestra la estructura del algoritmo de recocido simulado que se integró en el algoritmo GSTT.

Algoritmo 5.2 Recocido Simulado para el GSTT

1: Seleccionar una solución inicial i // Donde $f(i)$ es su función de costo
2: Seleccionar una temperatura final Tf // Criterio de paro
3: Seleccionar una temperatura inicial $To > Tf > 0$
4: Seleccionar un coeficiente $\propto$ de reducción de la temperatura To
5: Seleccionar un No. máximo de iteraciones ncM // ncM: No. de ciclos de Metrópolis
6: **Mientras** ($Tf < To$) // No alcanza criterio de paro **hacer:**
7: **Para** ($cM = 1$; $cM < ncM$; cM++) **hacer:**
8: Generar aleatoriamente una solución $j \in N(i)$
9: **Si** ($f(j) \leq f(i)$) **entonces:**
10: $i \Leftarrow j$.
11: **En otro caso, si** ($exp(-\frac{f(j)-f(i)}{To}) > random[0 \ldots 1)$) **entonces:**
12: $i \Leftarrow j$.
13: $To \Leftarrow \propto To$.

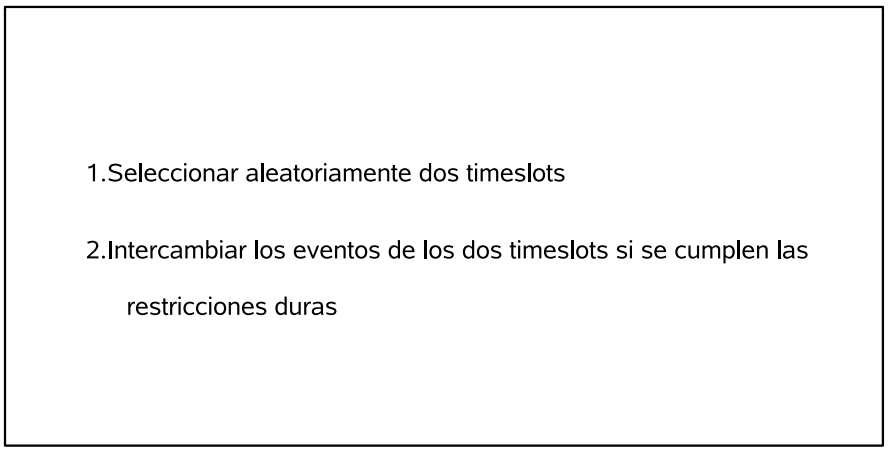
Al final, debido a que $Tf \geq To$, se alcanzó el criterio de paro y la mejor solución visitada por el algoritmo será la solución heurística almacenada en i.

5.2.1. Estructura de vecindad

Para que funcione correctamente el algoritmo de recocido simulado en la tarea de optimizar las soluciones que se hayan encontrado con el GSTT, es necesario incluir una perturbación que establezca los movimientos válidos durante el proceso de reacomodo de recursos. El reacomodo se debe realizar considerando la vecindad de la solución que se toma como referencia, a este proceso de reacomodo de recursos se le conoce como búsqueda local. Sin embargo, ya sabemos que el algoritmo de recocido simulado, debido al funcionamiento del algoritmo de metrópolis que tiene integrado y a las consideraciones de sintonía, tiene la cualidad de hacer búsquedas locales y en ocasiones salta de una vecindad a otra para no quedar atrapado en un óptimo local. Para realizar este reacomodo se requiere de un algoritmo de búsqueda local que preferentemente realice movimientos sencillos y que no tome mucho tiempo computacional en su ejecución, ya que en la mayoría de los casos, es necesario realizar muchos movimientos de reacomodo para encontrar mejores soluciones. Además de lo anterior, hay que considerar el tamaño del espacio de búsqueda que está relacionado directamente con la cantidad de elementos que formen la instancia problema. Entonces, al diseñar un algoritmo de búsqueda local que sirva como perturbación al recocido simulado se busca que tenga una estrategia operativa con movimientos sencillos para procesar el reacomodo de recursos y que no cargue al proceso con tiempos de computación muy altos.

En el algoritmo de SA la perturbación se genera en el ámbito de $N(i)$ y representa el entorno de una solución de referencia i, por esto $N(i)$ recibe el nombre de *Vecindad* ya que hace mención al conjunto de soluciones cercanas a la solución de referencia i. Por lo tanto, N es un conjunto de soluciones vecinas a una solución de referencia, que se pueden explorar haciendo una búsqueda de acuerdo a los movimientos incluidos en un algoritmo de búsqueda local.

La perturbación que se incluyó en el algoritmo de recocido simulado para hacer la búsqueda local de mejores soluciones, lo constituyen movimientos sencillos del tipo de *doble movimiento*. Esta perturbación se muestra en el algoritmo de búsqueda local del Cuadro 5.2 y es similar al algoritmo de doble movimiento que se presentó en la Sec-



Cuadro 5.2: Algoritmo de búsqueda local para el Recocido Simulado

ción 2.3.3. La implementación del algoritmo de búsqueda local se integró en el recocido simulado desde donde se llama con la instrucción: *Generar aleatoriamente una solución* $j \in N(i)$. El procedimiento básico de funcionamiento, es que del espacio de búsqueda, el algoritmo selecciona aleatoriamente dos timeslots y verifica si puede hacer una *permutación* de sus eventos. La permutación se realizará siempre y cuando al realizarlo se cumplen las restricciones duras; pero si al tratar de permutar los eventos no se cumplen las restricciones duras, se dejan los eventos en sus timeslots originales. Si la permutación se lleva a cabo se habrá obtenido una solución j que será muy cercana (vecina) a la solución i, ya que la distribución de eventos en los timeslots de las soluciones j e i solo difieren en esos dos eventos permutados. Recocido Simulado se encargará de evaluar la nueva solución j contra la solución de referencia i para escoger la mejor solución. Esta acción se repetirá hasta alcanzar el criterio de parada Tf

Donde el tamaño de la vecindad es:

$$N(j) = \frac{n_E(n_E - 1)}{2} \tag{5.1}$$

y donde n_E corresponde al número de eventos que componen la instancia problema.

5.2.2. Análisis de sensibilidad

La mayoría de los problemas se pueden plantear tanto como de *optimización* como de *satisfacción de restricciones*.

La diferencia entre los dos puede ser muy importante. Por ejemplo: Encontrar una ruta entre ciudades en el problema del agente viajero es trivial, encontrar la ruta más corta es NP.

Una idea fundamental dentro de los métodos heurísticos es que *lo más corto/barato es lo más rápido/mejor*

Entonces, si existe un criterio de aceptación (tolerancia) se habla de un problema de *semioptimización*. Cuando se habla de alta probabilidad, se tiene un problema de *optimización aproximada*.

La mayoría de los problemas son de semioptimización, ya que se establece un balance razonable entre la calidad de la solución y el costo de la solución. Algo un poco más elaborado es dotar a los algoritmos con parámetros ajustables para cambiar el compromiso entre calidad y costo.

Considerando los conceptos antes expuestos, con el algoritmo GSTT se buscó resolver el problema del UCTP con un planteamiento tanto de satisfacción de restricciones como de optimización y a través de la Estrategia de Asignación de Recursos que se estableció en el Capítulo 5.1.2. Se buscó también, que el método heurístico fuese corto/barato y a la vez se trató de balancearlo para que fuese rápido/mejor. En la optimización con recocido simulado, para tratar de establecer un balance razonable entre la calidad de la solución y el costo de la solución, se efectuó con cuidado el análisis de sensibilidad para establecer los parámetros de sintonía.

En la ejecución del algoritmo GSTT para probar el funcionamiento de mejora de soluciones con recocido simulado, se ejecutaron 100 pruebas para cada caso, matemáticamente se considera que 30 pruebas es una muy buena muestra estadística. De las 100 pruebas se escogieron las 20 mejores soluciones obtenidas y a cada solución el GSTT le aplicó el proceso de optimización con el recocido simulado.

El análisis de sensibilidad se aplicó utilizando como paradigma de instancia problema los dos benchmarks grandes que se conocen en la literatura como hard01.tim y hard02.tim. Las variables que se sometieron al análisis de sensibilidad fueron: la temperatura inicial To como parámetro de control, el coeficiente $\propto$ de reducción de la temperatura, el número máximo de iteraciones (número de ciclos de Metrópolis) ncM y la temperatura final Tf como criterio de paro. El rango establecido para cada una de estas variables se fraccionó de forma adecuada para obtener una secuencia de 20 valores espaciados uniformemente. Esta secuencia, así establecida, sirvió durante el análisis de sensibilidad para hacer un barrido con cada variable y obtener su valor de sintonía.

El rango de las variables del recocido simulado para realizar el análisis de sensibilidad, se establecieron en base a las recomendaciones que sobre el tema se encuentran en la literatura y a la experiencia técnica adquirida en el laboratorio de cómputo por el cuerpo académico Optimización y Software, adscrito al Centro de Investigación en Ingeniería y Ciencias Aplicadas (CIICAp) de la Universidad Autónoma del Estado de Morelos (UAEM), lugar donde se realizó el análisis de sensibilidad y las pruebas experimentales del algoritmo GSTT.

Para realizar el proceso del análisis de sensibilidad, primero se consideró una de las cuatro variables mencionadas y se ejecutó el GSTT para cada uno de los 20 valores de la secuencia establecida y para cada valor se pidieron 5 soluciones (20x5=100 pruebas por cada variable). Mientras se hizo el proceso de barrido con la primera variable, las otras 3 variables se dejaron con sus valores iniciales fijos -no sintonizados-. El parámetro de esta primera variable, que obtuvo la mejor solución, se escogió y fijó como valor de sintonía. Posteriormente, se tomó la segunda variable y también se hizo el proceso de barrido con

los 20 valores y el pedido de 5 soluciones (20 valores x 5 soluciones). Durante la ejecución de las 100 pruebas con esta segunda variable, se dejaron fijas las otras 3 variables, la 1ª variable con el valor de sintonía antes obtenido y las variables 3ª y 4ª con sus valores iniciales aun no sintonizados. El valor de esta segunda variable que obtuvo la mejor solución se escogió como valor de sintonía. Estos pasos se repitieron para la tercera y cuarta variable.

El orden para realizar el análisis de sensibilidad, se aplicó estrictamente con la secuencia establecida en los cuatro pasos que se dan en la lista a continuación. Los nombres en itálica son aquellos que normalmente se les da en el ámbito de la optimización/computación.

1. Temperatura Inicial To o *Parámetro de control*. El caso práctico recomienda iniciar con 5 unidades y disminuir hasta llegar a 2 unidades. Entonces $((5-2)/19 = 0.16)$, las divisiones fueron de 0.16. Sin embargo, este valor de To se puede obtener también según la teoría mostrada en el artículo [Sanvicente-Sanchez and Frausto, 2004].

2. Número máximo de iteraciones ncM o longitud de la *cadena de Markov* (ciclos de Metrópolis) para alcanzar el criterio de equilibrio térmico. Se recomienda iniciar con 3,000 unidades e ir disminuyendo hasta llegar a 200. Entonces $((3000-200)/19 = 147.4)$, las divisiones serán de 147.4.

3. Función de reducción de la temperatura $\propto$ o *coeficiente de control*. Se utiliza para decrementar la temperatura To, por lo tanto deberá ser menor a 1. Iniciar con 0.988 e ir disminuyendo en pasos de 0.001 en 0.001 hasta llegar a 0.97. Entonces $((0{,}988-0{,}97)/19 = 0.001)$, las divisiones serán de 0.001.

4. *Temperatura Final* Tf como criterio de paro. Iniciar con 10 unidades e ir disminuyendo hasta llegar a la unidad. Entonces $((10-1)/19 = 0.5)$, las divisiones serán de 0.5.

Al final, los parámetros de sintonía de las cuatro variables (To, ncM, $\propto$ y Tf) serán aquellos en donde se hayan obtenido los *mejores valores de optimización*.

Capítulo 6

Análisis de Resultados

El propósito de este capítulo es la presentación de los resultados obtenidos en las pruebas efectuadas al algoritmo GSTT. Con este fin, inicialmente se muestra el formato de las instancias (benchmarks) que sirven de datos de entrada. Posteriormente se explica, el proceso para obtener las soluciones y los parámetros de sintonía obtenidos por el análisis de sintonía. Para concluir, se exponen las pruebas experimentales que se obtuvieron con los 5 algoritmos de la teoría y el GSTT. También se presenta un análisis de la efectividad de los resultados obtenidos con las instancias medianas y grandes. Para terminar se hace la comparación de los resultados obtenidos por el GSTT contra los resultados que se obtuvieron de los algoritmos representativos que se encuentran en la literatura.

6.1. Formato de Instancias

Como ya se trató en la Sección 2.6 existen en la literatura tres clases de instancias problema o benchmarks, los cuales se pueden utilizar como paradigmas para hacer las pruebas de funcionamiento de los algoritmos que se implementen para programar los recursos escolares. De esta forma, también se estará en posibilidad de hacer comparación de los resultados que se obtengan contra los resultados que obtiene de los algoritmos representativos en la literatura. La clasificación de las instancias se hizo de acuerdo a la cantidad de recursos con que se estructuraron y reciben el nombre de: *pequeña, mediana y grande* instancia. Cada instancia esta compuesta por 8 diferentes tipos de recursos. Las tres clases de instancias problema poseen la misma estructura; pero diferente cantidad

de recursos. Aunque cada tipo de recurso es diferente (salones, No. de alumnos, No. de eventos, etc), la cantidad de recursos es la misma para cada clase de instancia, entonces, instancias de la misma clase contienen la misma cantidad de recursos. Lo que hace diferente a una instancia de otra de la misma clase son, por ejemplo: las necesidades de cada evento (proyector, pintarron, Etc.), las características del recurso, el tamaño de los salones, las materias asignadas a cada alumno, Etc. Los parámetros y tipos de recursos que componen las instancias problema se muestran en el Cuadro 5.1.

Para el algoritmo GSTT, también se utilizaron los benchmarks como datos de entrada, tanto para verificar su funcionamiento durante la implementación, como para el proceso de pruebas para verificar los resultados y comparar la eficacia y la eficiencia.

Para la ejecución del algoritmo GSTT, los datos de entrada o instancias problema se deberán encontrar almacenados en archivos con el formato adecuado para que puedan ser leídos por el GSTT. Los datos en las instancias problema o benchmarks están organizados con el formato que se muestra a continuación (todos los números son enteros y las entrelíneas son espacios).

- **Primera línea:** El número de eventos, número de salones, número de características, número de estudiantes.
- **Una línea para cada salón:** Cada línea indicará el aforo del salón.
- **Una línea para cada estudiante/evento:** Con un cero o uno. Cero indica que el estudiante no atiende el evento; uno indica que el estudiante atiende el evento. Por ejemplo, si hubiera 3 estudiantes y 4 eventos programados entonces lo siguiente:

  0
  1
  0
  0
  1
  1

0
0
0
0
1
0

Representará la siguiente matriz de incidencia de: *eventos por estudiante*

$$\begin{array}{cc} & evento \\ & \begin{array}{cccc} 0 & 1 & 0 & 0 \end{array} \\ estudiante & \begin{array}{cccc} 1 & 1 & 0 & 0 \end{array} \\ & \begin{array}{cccc} 0 & 0 & 1 & 0 \end{array} \end{array}$$

Que indicará:
El primer estudiante atiende el segundo evento
El segundo estudiante atiende el primer y segundo eventos
El tercer estudiante atiende el tercer evento

- **Una línea para cada salón/característica:** Un cero si el salón no satisface la característica requerida, o un uno si el salón si satisface la característica requerida. Por ejemplo, si hubiera 3 salones y 4 características entonces lo siguiente:

0
1
0
0
1
1
0
0
0
0

1

0

Daría la siguiente matriz de incidencia de: *características por salón*

$$\begin{array}{cc} & \textit{característica} \\ & \begin{matrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{matrix} \\ \textit{salón} & \end{array}$$

Que indicará:
El primer salón satisface la segunda característica
El segundo salón satisface la primera y segunda característica
El tercer salón satisface la tercera característica

- **Una línea para cada evento/necesidad**: Un cero si el evento no requiere ese necesidad, o un uno en caso de si requerir la necesidad. Por ejemplo, si hubiera 3 eventos y 4 necesidades entonces lo siguiente:

0

1

0

0

1

1

0

0

0

0

1

0

Daría la siguiente matriz de incidencia de: *necesidades por evento*

$$
\begin{array}{cc}
 & necesidad \\
 & \begin{array}{cccc} 0 & 1 & 0 & 0 \end{array} \\
evento & \begin{array}{cccc} 1 & 1 & 0 & 0 \end{array} \\
 & \begin{array}{cccc} 0 & 0 & 1 & 0 \end{array}
\end{array}
$$

Que indicará:

El primer evento requiere la segunda necesidad

El segundo evento requiere la primera y segunda necesidad

El tercer evento requiere la tercera necesidad

6.2. Benchmarks y Equipo de Pruebas

Para realizar las pruebas se seleccionaron los benchmarks creados por el grupo Metaheuristic Network y estan disponibles en [http://www.metaheuristic.org/, 2009] o también en [http://iridia.ulb.ac.be/Supp/IridiaSupp2002-001/index.html, 2009]. Como es bien sabido en la comunidad científica, los benchmarks son instancias problema que se utilizan como paradigma para evaluar el desempeño de los algoritmos que se crean en la comunidad y poder hacer comparaciones con los resultados que se obtengan en las pruebas. En la investigación que se reporta en este trabajo doctoral, también se utilizaron estos benchmarks para realizar las pruebas de funcionamiento y poder comparar los resultados obtenidos contra los resultados reportados en la literatura.

Como ya se expuso en la sección 5.1.1, los benchmarks existentes en la literatura están divididos en tres clases que son: pequeños, medianos y grandes. Esta división es de acuerdo con la cantidad de recursos que propone la instancia problema y que directamente afecta la complejidad para obtener una solución en tiempo polinomial. Los benchmarks pequeños son cinco y los medianos también son cinco; los benchmarks grandes solo son dos.

Para las pruebas del GSTT, se corrieron pruebas con benchmarks medianos y grandes, no se corrieron pruebas con los benchmarks pequeños, ya que se considero irrele-

Recursos de instancia	mediana	grande
Número de eventos	400	400
Número de salones	10	10
Número de características	5	10
Características aproximadas por salón	3	5
Porcentaje de características usadas	80	90
Número de estudiantes	200	400
Número máximo de eventos semanal por estudiante	20	20
Número máximo de estudiantes por evento	50	100

Cuadro 6.1: Recursos que contienen los benchmarks medianos y grandes

vante correr pruebas para estos benchmarks en virtud de que si el algoritmo implementado puede obtener soluciones factibles con los benchmarks medianos, los podrá obtener sin dificultad con benchmarks pequeños. Además, lo que se busca es poder obtener soluciones de los benchmarks grandes, ya que es el objetivo más preciado como meta cuando se ataca la problemática de programación -o calendarización- de recursos. Los nombres de los benchmarks medianos son: medium01.tim, medium02.tim, medium03.tim, medium04.tim y medium05.tim. Los nombres de los benchmarks grandes son: hard01.tim y hard02.tim. Los recursos que componen los benchmarks medianos y grandes se muestran en el Cuadro 6.1.

Para todas las pruebas experimentales se utilizó el mismo equipo de cómputo, el cual tiene las siguientes características: CPU Pentium con velocidad de 2.8 GHz. con sistema operativo Linux.

El lenguaje que se utilizó para programar el algoritmo GSTT es el Lenguaje gcc, que viene incluido como software libre en el sistema operativo Linux.

6.3. Parámetros Sintonizados

Después de efectuar el análisis de sensibilidad, como se indica en la Sección 5.2.2, se obtuvieron los parámetros de sintonía para las variables del recocido simulado encargado de optimizar las soluciones factibles que se obtengan. Los parámetros obtenidos se incluyeron y fijaron en el código correspondiente del algoritmo de recocido simulado, el

cual a su vez esta integrado en el código del programa del GSTT.

Para obtener los parámetros sintonizados se requirió la ejecución exhaustiva del algoritmo GSTT con el fin de obtener una buena sintonía (muestra) de cada parámetro. El proceso del análisis de sensibilidad se llevo a cabo tomando como datos de entrada los dos benchmarks clasificados como grandes. La ejecución para cada benchmark tomó un tiempo polinómico; pero como se ejecutaron 100 pruebas para obtener cada uno de los parámetros, el tiempo acumulado de proceso para obtener la sintonización completa se llevó varios días.

Las variables de la metaheurística de recocido simulado que se sometieron al análisis de sensibilidad fueron: la temperatura inicial To como parámetro de control, el coeficiente $\propto$ para la reducción de la temperatura, el número máximo de iteraciones ncM y la temperatura final Tf como criterio de paro. Los parámetros de las variables para el análisis de sensibilidad, se establecieron en base a las recomendaciones que sobre el tema se encontraron en la literatural y también a la experiencia técnica desarrollada en el laboratorio de cómputo por el cuerpo académico Optimización y Software adscrito al Centro de Investigación en Ingeniería y Ciencias Aplicadas (CIICAp) de la Universidad Autónoma del Estado de Morelos (UAEM).

El criterio de hacer un barrido con cada una de las variables del análisis de sensibilidad, es *obtener los parámetros de sintonía* con los que se pueden obtener mejores soluciones que sean las más cercanas a la solución óptima sin descartar el obtenerla. Los parámetros de sintonía que se obtuvieron al aplicar el análisis de sensibilidad para las instancias problema grandes hard01 y hard02 fueron:

1. La temperatura inicial To es: 3.24 para hard01 y 2.28 para hard02.
2. El número máximo de iteraciones (número de ciclos de Metrópolis) ncM para cada valor fijado de temperatura To es: 2852 para hard01 y 1626 para hard02.
3. El coeficiente $\propto$ para reducir la temperatura To en cada recorrido de la cadena ncM resultó de: 0.984170 para hard01 y 0.982318 para hard02.

Variable	hard01	hard02
Temperatura inicial, *To*	3.24	2.28
Número máxima de iteraciones, *ncM*	2852	1626
Coeficiente α de reducción de la temperatura *To*	0.98417	0.982318
Temperatura final, *Tf*	1	1

Cuadro 6.2: Resumen de los parámetros de sintonía

4. La temperatura final Tf para terminar el proceso de recocido simulado resultó de: 1 para hard01 y para hard02.

En el Cuadro 6.2 se muestra una tabla con el resumen de los parámetros de sintonía que se mencionan en la lista anterior. Los parámetros de sintonía obtenidos del análisis de sensibilidad se integran en la metaheurística de Recocido Simulado. En nuestro caso, considerando que se hicieron las pruebas para los dos benchmarks, hard01 y hard02, que se encuentran en la literatura y que son en la actualidad las instancias más grandes existentes, se podrían dejar fijos en el GSTT los parámetros de sintonía que se obtuvieron con la instancia hard01, ya que al analizar los resultados obtenidos (Capítulo 6.5) en las pruebas realizadas a ambas instancias grandes, se deduce que la instancia hard01 tiene mayor complejidad computacional.

En el Cuadro 6.6 del Capítulo 6.5 se presenta una tabla con los resultados del proceso de análisis de sensibilidad. Los mejores parámetros de sintonía aparecen marcados en tono rojizo/sombreado excepto el parámetro de criterio de parada Tf. Del criterio de parada se hicieron 2 corridas extras con parámetros de Tf cercanos al valor donde la FO fué más pequeña. Esto, con el fin de conocer si aun se presentaba mejora.

6.4. Pruebas Experimentales

Para obtener un panorama más objetivo y amplio en la comparación de resultados, en el laboratorio de cómputo del CIICAp-UAEM se ejecutaron los algoritmos de las metaheurísticas ACO, ILS, GA, SA y TS que se obtuvieron del grupo Metaheuristic Network con el fin de tratar de reproducir los resultados que reportan en la literatura. Los

algoritmos fuentes y ejecutables se obtuvieron de: [http://www.metaheuristic.org/, 2009] y [http://iridia.ulb.ac.be/Supp/IridiaSupp2002-001/index.html, 2009] que son sitios del grupo Metaheuristic Network. Los algoritmos obtenidos de estos sitios se trataron de ejecutar; pero no fue posible. Entonces se trató de compilar los fuentes para tratar de obtener el ambiente adecuado. Inicialmente se tuvieron muchos problemas para lograr el objetivo, para remediarlo, se contactó directamente con el grupo Metaheuristic Network en particular con el Dr. Max Manfrin en diversas ocasiones para preguntar y conocer las características necesarias para compilar y ejecutar los algoritmos. Algunas de las necesidades principales fueron que, para poder ejecutar los algoritmos se requería necesariamente la versión idéntica del sistema operativo Linux donde compilaron los algoritmos (g++ V2.95, Linux 7), ya que el código ejecutable utiliza determinadas librerías exclusivas de la versión que utilizó el grupo Metaheuristic Network para compilar y ejecutar sus algoritmos. Otra característica importante para la ejecución fue que, los algoritmos no obtienen soluciones con cualquier semilla, para lograrlo hay que dar por línea de comandos la semilla específica. Afortunadamente después de solventadas varias incertidumbres, incluyendo las anteriores, se logró establecer en el laboratorio de cómputo del CIICAp-UAEM el ambiente adecuado para la compilación y ejecución de los algoritmos.

Como datos de entrada para realizar las pruebas de los algoritmos ACO, ILS, GA, SA y TS creados por el grupo Metaheuristic Network y para el algoritmo GSTT creado en el CIICAp-UAEM, se utilizaron como problemas a resolver los benchmarks de las cinco instancias medianas y las 2 instancias grandes. Para la ejecución de cualquiera de estos algoritmos, es necesario que la instancia problema de interés esté alojada en un archivo de donde los algoritmos la tomarán como datos de entrada y los procesarán para tratar de obtener una solución.

Para arrancar la ejecución de los algoritmos ACO, ILS, GA, SA, TS y para GSTT, por línea de comandos se deberá especificar la información necesaria para la ejecución. Un ejemplo de esta información es la siguiente:

Semilla = 3

Función objetivo	Primer Sol. Tiempo Seg	Sol.mejorada Tiempo Seg	T. Total en Seg.	Eventos en Último periodo	Violaciones de S1	S2	S3
1059	1783	10905	12688	2	55	914	90
1376	1843	13595	15438	4	108	1165	103
1149	1380	9321	10701	2	43	1006	100
988	1147	9205	10352	2	40	864	84
1166	1804	15081	16885	3	85	985	96

Solución optimizada por Recocido Simulado = 80
Tiempo de ejecución de Recodido Simulado = 47311 segundos
Archivo: resultadoSemilla3T3.24ncm2852.6c0.98417Tf10-SintoniaGSTT12xHard01.txt

Cuadro 6.3: Muestra del resumen en el archivo de salida para el GSTT

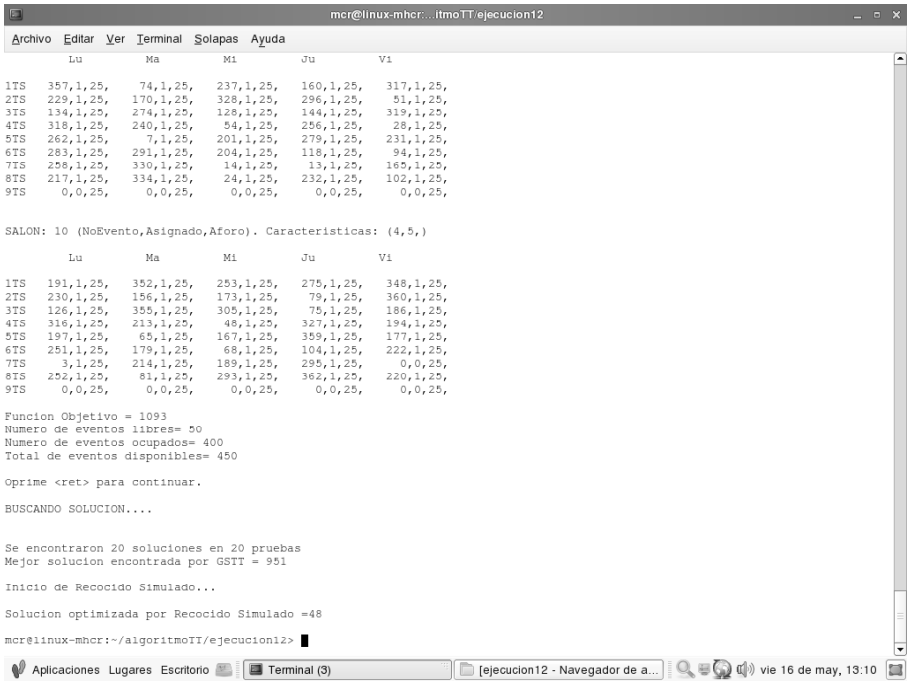
/home/tabladehorarios.aco -i /home/tabladehorarios/instancias/hard01.tim
-o /home/tabladehorarios/salidas/solucion.aco.9000.20.314.hard01
-n 20 -s 314 -t 9000

donde: se ejecuta para el algoritmo ACO, la instancia de entrada es hard01.tim, la solución se almacenará en el archivo solucion.aco.9000.20.314.hard01, el número de pruebas solicitado es 20 (-n 20), la semilla es 314 (-s 314) y el tiempo límite en segundos de cada prueba se pide de 9000 máximo (-t 9000).

Además, para el GSTT se guarda un resumen con los parámetros de las soluciones obtenidas (según la cantidad de soluciones solicitadas) y de la mejor solución optimizada. Algunos de los parámetros que presentará el archivo de salida con el resumen serán: valor de las funciones objetivo, tiempos de ejecución y violaciones de restricciones suaves. De la solución optimizada mostrará: Valor de la función objetivo y el tiempo consumido durante la optimización. En el Cuadro 6.3 se muestra un ejemplo del resumen que se guarda en archivo de salida.

El desarrollo de ejecución de la prueba, por omisión se presenta por pantalla, el proceso transcurre de acuerdo a la ejecución de las actividades programadas en el Algoritmo 5.1 GSTT y Figura 5.2. Primero mostrará el proceso de búsqueda de la solución parcial factible con indicación de los eventos que están teniendo problema para encontrar ubicación en los timeslots disponibles hasta que logra obteanalisisEfectividadner una so-

lución parcial factible. Posteriormente, los eventos remanentes en ENA2 los reasigna en timeslots vacíos de TT sin verificar si al hacerlo se cumplen las restricciones duras, como lo más seguro es que la solución se vuelve infactible, se ejecutan las actividades 10, 11 y 12 para tratar de reencontrar una solución factible. Durante este proceso, el GSTT presenta por pantalla los eventos con los que esta teniendo dificultad para asignarlos en timeslots adecuados. Si encuentra una solución factible, entonces muestra en pantalla un resumen con el valor de la función objetivo y el tiempo que tardo en obtenerla; si no encuentra solución factible, escribe por pantalla un texto indicando que no encontró solución de tal prueba. En ambos casos, de haber obtenido solución factible o no, envía también al archivo de salida el resumen que mostró por pantalla y continuará el proceso ya descrito, hasta completar de obtener el número de soluciones factibles que se le hayan solicitado por línea de comandos. Al terminar de obtener el GSTT el número de soluciones solicitadas, verifica cual de ellas es la mejor (menor valor de FO) y la selecciona como mejor solución i obtenida, y que sirve de entrada para ejecutar el proceso de optimización con recocido simulado. Normalmente, el proceso de optimización es el que consume el mayor tiempo de computación. Cuando termina el proceso de optimización, por pantalla se presenta la solución obtenida, representada por las tablas de horarios, con una tabla por salón donde se mostrarán (la mayoría en código numérico) entre otros, los siguientes datos: el número de salón, las características/necesidades del salón/evento, el número de evento, asignación (1 si, 0 no), aforo del salón y un resumen de los parámetros de la solución factible que se obtuvo antes de pasar al proceso de optimización, como por ejemplo: el parámetro de la Función Objetivo. En el Cuadro 6.4 se puede ver un ejemplo de este resultado como una Tabla de Horarios. Además, el GSTT presenta por pantalla, el resumen de el tiempo de optimización y el valor optimizado de la función objetivo con el texto de: “Solución optimizada por Recocido Simulado = valor”. Este resumen también se envía al archivo de salida, ver muestra en el Cuadro 6.3.



Cuadro 6.4: Presentación por pantalla de una solución (tabla de horarios)

6.5. Análisis de Efectividad de los Resultados

En este capítulo se hace un análisis y comparación de la efectividad que obtiene el GSTT para obtener soluciones factibles en las pruebas efectuadas. Este análisis y comparación se hace a partir de los resultados obtenidos con la ejecución en el laboratorio de computación del CIICAp-UAEM del algoritmo GSTT y de los algoritmos ACO, GA, ILS, SA y TS.

6.5.1. Efectividad con problemas medianos

La obtención de soluciones factibles de instancias medianas (o benchmarks medianos) para el GSTT no es gran dificultad, de las 50 pruebas realizadas para cada una de las 5 instancias se obtuvieron soluciones satisfactorias en todas las pruebas (250 pruebas), es decir tiene una efectividad del 100 %.

Respecto a la evaluación de la Eficacia, considerando la calidad de las soluciones (número de restricciones suaves violadas) el GSTT se coloca en segundo lugar, después

de Recocido Simulado (SA). La comparación de los resultados contra los otros algoritmos se verá en el Cuadro 6.8.

En el Cuadro 6.5 se muestran las mejores soluciones obtenidas con GSTT de instancias medianas. Para cada uno de los cinco benchmarks se muestra en una columna la mejor solución sin aplicar optimización, el rango de las soluciones va de 436 a 478 violaciones de restricciones suaves y las soluciones las obtiene tomando desde medio segundo hasta un valor de 17.9 seg que consume para Medium03; sin embargo para Medium05 la eficiencia es de 73.9 seg que es relativamente alto, considerando el resto de los valores. Respecto a la efectividad de la optimización de GSTT con recocido simulado para estas instancias, es notorio que obtiene muy buenos resultados de Función Objetivo, que va de 91 a 121 con tiempos para la optimización de 602 seg hasta 1,097.9 seg que equivale a menos de 20 minutos. Por lo tanto, los resultados obtenidos por el GSTT con instancias medianas son satisfactorios.

El objetivo de la investigación doctoral desde el inicio fue tratar de obtener soluciones satisfactorias de las instancias grandes (benchmarks grandes) y que se pudieran obtener en tiempo polinómico. Sin embargo, también se hicieron pruebas con las instancias medianas para verificar si no se presentaban dificultades para tratar esos problemas. Así que el GSTT no tiene problemas con estas instancias y se constató que tiene una efectividad del 100 % para obtener soluciones factibles de instancias medianas. Los algoritmos que se encuentran en la literatura, la mayoría obtiene soluciones satisfactorias de instancias medianas, aunque no necesariamente al 100 %. En el Cuadro 6.8 se podrá apreciar un comparativo de las ejecuciones realizadas en el laboratorio de computación del CIICAp-UAEM con los algoritmos que se encuentran en el estado del arte y de los cuales contamos con los códigos ejecutables.

6.5.2. Efectividad con problemas grandes

Las pruebas realizadas con el GSTT para tratar de obtener soluciones satisfactorias de instancias grandes (benchmarks grandes) fueron de 100 pruebas de cada benchmark y se sometieron a optimización las 20 mejores soluciones.

MEJORES SOLUCIONES OBTENIDAS CON GSTT DE INSTANCIAS MEDIANAS

50 pruebas con cada instancia

VALOR DE FUNCIÓN OBJETIVO Y TIEMPO (Eficacia y Eficiencia)

Instancias	*Mejor Solución*	*Sol. Optimizada*	*Tiempo total*
Medium01	478 en 7 seg	121 en 939.8 seg	946.8 seg
Medium02	475 en 0.5 seg	111 en 1,097.9 seg	1,098.4 seg
Medium03	456 en 17.9 seg	178 en 963 seg	980.9 seg
Medium04	436 en 1.8 seg	91 en 1,060 seg	1,061.8 seg
Medium05	450 en 73.9 seg	120 en 602 seg	675.9 seg

Pruebas efectuadas en Laboratorio de Cómputo del CIICAp-UAEM
en equipo: CPU Pentium 2.8 GHz, S.O. Linux.

Cuadro 6.5: Mejores soluciones obtenidas por GSTT de instancias medianas

Para hacer un análisis de los resultados obtenidos, se comenzará por presentar el proceso del análisis de sensibilidad que se realizó para los dos benchmarks existentes en la literatura y que se toman como paradigmas para todos los que implementan algoritmos con la finalidad de encontrar soluciones del UCTP. En las tablas del Cuadro 6.6 se muestran los resultados del análisis de sensibilidad del GSTT con la instancia hard01 y en las tablas del Cuadro 6.7 se muestran los resultados del análisis de sensibilidad del GSTT con la instancia hard02. En estas tablas, los parámetros de sintonía se encuentran en rojo/sombreado, excepto para el parámetro de sintonía Tf.

Para encontrar el parámetro de sintonía del Criterio de Parada Tf o Temperatura Final con hard01, se tuvieron que ejecutar 3 conjuntos de procesos más y para hard02, 2 conjuntos de procesos más. Esto con el fin de fijar más adecuadamente el parámetro Tf con el que recocido simulado podría optimizar mejor.

La efectividad de GSTT para obtener soluciones satisfactorias es bastante buena si no excelente, ya que durante las pruebas, de cada 20 pruebas puede obtener 19 soluciones satisfactorias. Se toma este parámetro de 20 pruebas como patrón para conocer la efectividad para obtener soluciones satisfactorias, en razón de ser congruente con el rango utilizado en la literatura. En particular con el grupo Metaheuristic Network que evalúa con 20 pruebas para cada caso.

Es importante resaltar que durante el proceso de análisis de sensibilidad, como se puede observar en las tablas de los Cuadros 6.6 y 6.7, que desde que se aplica la primera etapa de optimización, con el barrido de valores de la Temperatura Inicial To para encontrar este parámetro de sintonía, los resultados de la Función Objetivo no varían durante todo el resto del proceso de análisis de sensibilidad, excepto cuatro valores para hard01, que por cierto son los valores más elevados y son los que sólo se optimizan; pero el resto permanece fijo. Con la instancia hard02, dos valores, también los más altos son los únicos que se optimizan y el resto igualmente permanecen fijos. Lo que podemos deducir de este análisis, es que definitivamente la metaheurística de *Recocido Simulado no es adecuada para optimizar problemas del UCTP*. También se puede deducir, que el recocido simulado al tratar con estos problemas de calendarización, puede encontrar buenas soluciones a temperaturas altas; pero a temperaturas muy bajas le cuesta trabajo obtener mejores soluciones, aunque las sigue obteniendo (con las 4 soluciones que optimiza para hard01 y las 2 soluciones para hard2); pero el tiempo de computación se incrementa ya que toma más tiempo mejorar una solución y este costo computacional disminuye la eficiencia. En fin, aun hay bastante campo en esta área para continuar investigando.

6.6. Comparación con otras metaheurísticas

En el Cuadro 6.8 se muestra una tabla comparativa con las mejores soluciones optimizadas de instancias medianas. Estos resultados se obtuvieron también en el laboratorio de computación del CIICAp-UAEM y se ejecutaron 50 pruebas con cada una de las 5 instancias para cada uno de los algoritmos ACO, GA, ILS, SA, TS y GSTT (50x5x6=1500

GSTT con Recocido Simulado
Análisis de Sensibilidad
Pág. 1

No. de Solución	Temp. Inic. To	Mejor Solución FO	Mejor Solución Tiempo en Seg	Solución Optimizada con RS FO	Solución Optimizada con RS Tiempo en Seg
1	5	988	10256	80	3642
2	4.84	1055	12593	892	8843
3	4.68	954	11106	104	3885
4	4.52	1043	11961	114	5026
5	4.36	1062	10239	58	8071
6	4.2	962	13272	116	3917
7	4.04	1017	9225	54	6917
8	3.88	1062	12572	156	3119
9	3.72	1080	9667	118	4003
10	3.56	978	10529	58	6619
11	3.4	1052	10718	873	8811
12	3.24	1006	9252	52	9000
13	3.08	1041	7712	763	10019
14	2.92	1003	8080	818	16841
15	2.76	1011	10676	56	6748
16	2.6	1055	12365	58	5254
17	2.44	1039	15686	120	1099
18	2.28	974	12979	56	6159
19	2.12	968	9930	60	8939
20	1.96	943	10291	56	8713

GSTT con Recocido Simulado Cont...
Análisis de Sensibilidad
Pág. 2

No. de Solución	ncM No.Máx.Iterac	Solución Óptima con RS FO	Solución Óptima con RS Tiempo en Seg	α Coef. Reduc.Temp	Solución Óptima con RS FO	Solución Óptima con RS Tiempo en Se
1	3000	80	46710	0.988800	80	16723
2	2852.6	530	72669	0.988337	439	139100
3	2705.2	104	35941	0.987874	104	79572
4	2557.8	114	37326	0.987411	114	79914
5	2410.4	58	50025	0.986948	58	10350
6	2263	116	27717	0.986485	116	59405
7	2115.6	54	43079	0.986022	54	98635
8	1968.2	156	25804	0.985559	156	64813
9	1820.8	118	26498	0.985096	118	53106
10	1673.4	58	37931	0.984633	58	90254
11	1526	671	50979	0.984170	568	128235
12	1378.6	52	34050	0.983707	52	51546
13	1231.2	656	48771	0.983244	594	124987
14	1083.8	619	51537	0.982781	584	128995
15	936.4	56	23415	0.982318	56	84215
16	789	58	20473	0.981855	58	81576
17	641.6	120	9780	0.981392	120	53862
18	494.2	56	2298	0.980929	56	79496
19	346.8	60	2176	0.980466	60	83573
20	199.4	56	1793	0.980003	56	57110

Sintonía de GSTT con Recocido Simulado Cont...
Análisis de Sensibilidad
Pág.3

No. de Solución	Tf Temp. Fina	Solución Optima con RS FO	Solución Optima con RS Tiempo en Seg	Tf Temp. Fina	Solución Optima con RS FO	Solución Optima con RS Tiempo en Se	Tf Temp. Fina	Solución Optima con RS FO	Solución Optima con RS Tiempo en Se
1	10	80	47311	1.9	80	62274	1	80	67093
2	9.5	1055	95637	1.9	692	113220	1	577	127184
3	9	104	48283	1.9	104	62252	1	104	67330
4	8.6	114	49682	1.9	114	63361	1	114	70993
5	8.1	58	75784	1.9	58	91841	1	58	96183
6	7.6	116	39601	1.9	116	49068	1	116	52775
7	7.2	54	73828	1.9	54	88452	1	54	96505
8	6.7	156	47981	1.9	156	58178	1	156	58494
9	6.2	118	51318	1.9	118	62336	1	118	62458
10	5.7	58	75605	1.9	58	89388	1	58	98779
11	5.3	904	97761	1.9	680	111513	1	568	117065
12	4.8	52	82617	1.9	52	93745	1	52	100284
13	4.3	865	100406	1.9	636	112381	1	534	119495
14	3.8	849	110036	1.9	652	122295	1	523	134032
15	3.4	56	78939	1.9	56	87252	1	56	97202
16	2.9	58	78040	1.9	58	85636	1	58	88620
17	2.4	120	54598	1.9	120	59517	1	120	64783
18	1.9	56	79402	1.9	56	85923	1	56	90926
19	1.5	60	90871	1.9	60	94362	1	60	99760
20	1	56	93199	1.9	56	94684	1	56	102791

Cuadro 6.6: Parámetros de Sintonía y Soluciones de instancia hard01 con GSTT

Sintonía de GSTT con Recocido Simulado

Análisis de Sensibilidad

Pág. 1

Solución	Temp. Inic.	Mejor	Solución	Solución Óptima con RS		*ncM*	Solución Óptima con RS	
	To	FO	Tiempo en Seg	FO	Tiempo en Seg	No.Máx. Iterac	FO	Tiempo en Seg
1	5	1020	9371	42	1008	3000	42	51928
2	4.84	1057	8225	46	7344	2852.6	46	42821
3	4.68	1075	12245	148	4822	2705.2	148	33023
4	4.52	965	8677	92	5816	2557.8	92	37086
5	4.36	1023	14914	138	3030	2410.4	138	30604
6	4.2	996	9248	96	6237	2263	96	37191
7	4.04	1003	9336	48	3545	2115.6	48	35037
8	3.88	950	6665	44	1501	1968.2	44	41147
9	3.72	1053	7146	66	7107	1820.8	66	35330
10	3.56	1103	8684	42	7343	1673.4	42	37424
11	3.4	1024	9976	690	11451	1526	545	46320
12	3.24	979	6827	40	5431	1378.6	40	31570
13	3.08	991	9675	90	5026	1231.2	90	20666
14	2.92	1035	7884	96	5774	1083.8	96	20568
15	2.76	1058	7982	669	17815	936.4	604	47143
16	2.6	1028	8671	90	2347	789	90	12969
17	2.44	945	1155	40	902	641.6	40	14511
18	2.28	994	7295	36	8411	494.2	36	14783
19	2.12	1008	10655	48	6288	346.8	48	10335
20	1.96	1057	11626	96	1796	199.4	96	3761

Sintonía de GSTT con Recocido Simulado Cont.

Análisis Sensibilidad

Pág.2

Solución	α Coef	Solución Óptima con RS		*Tf*	Solución Óptima con RS		*Tf*	Solución Óptima con RS	
	Reduc. Temp.	FO	Tiempo en Seg	Temp. Final	FO	Tiempo en Seg	Temp. Final	FO	Tiempo en Seg
1	0.988800	42	62278	10	42	35302	1	42	47797
2	0.988337	46	54530	9.5	46	31869	1	46	43688
3	0.987874	148	37127	9	148	18775	1	148	25830
4	0.987411	92	41647	8.6	92	22181	1	92	30578
5	0.986948	138	35247	8.1	138	20199	1	138	28195
6	0.986485	96	44585	7.6	96	26790	1	96	35940
7	0.986022	48	46354	7.2	48	29353	1	48	39071
8	0.985559	44	51954	6.7	44	34051	1	44	45235
9	0.985096	66	33835	6.2	66	29015	1	66	37579
10	0.984633	42	48744	5.7	42	33930	1	42	44291
11	0.984170	484	62099	5.3	1024	44729	1	529	53160
12	0.983707	40	45295	4.8	40	32129	1	40	42033
13	0.983244	90	32798	4.3	90	24593	1	90	31032
14	0.982781	96	34957	3.8	96	27408	1	96	33276
15	0.982318	471	68121	3.4	757	52760	1	471	60785
16	0.981855	90	28900	2.9	90	31679	1	90	39730
17	0.981392	40	4878	2.4	48	4738	1	48	34338
18	0.980929	36	44403	1.9	36	37987	1	36	45482
19	0.980466	48	41671	1.5	48	35386	1	48	42776
20	0.980003	96	20013	1	96	25558	1	96	30324

Cuadro 6.7: Parámetros de Sintonía y Soluciones de instancia hard02 con GSTT

pruebas) para obtener una tabla comparativa con las mejores soluciones factibles optimizadas de cada algoritmo en cada instancia mediana.

De los resultados en el Cuadro 6.8 se puede observar que SA obtiene los mejores valores (calidad). Es decir, soluciones más eficaces con Función Objetivo menor (71, 70, 111, 65 y 88 violaciones de restricciones suaves). GSTT se sitúa en segundo lugar, ya que de las 5 instancias obtiene 3 segundas mejores soluciones eficaces optimizadas (111, 91 y 120 violaciones de restricciones suaves); los otros dos segundos lugares los obtiene ILS con las instancias Medium01 y Medium03 (120 y 171 respectivamente). Los mejores tiempos (eficiencia) los obtiene TS en 3 de las 5 instancias: medium01, 253 seg; medium04, 508 seg y medium05, 102 seg. En segundo lugar GA con 2 mejores tiempos: medium02, 638 seg y medium03, 567 seg. Estos resultados indican que SA es excelente para obtener soluciones de instancias medianas para el UCTP, como ya se sabía en la revisión de la literatura. El GSTT se diseñó con la finalidad de obtener soluciones eficaces de instancias grandes utilizando a SA para optimizarlas; sin embargo se sitúa en segundo lugar al obtener 3 de 5 segundas mejores soluciones eficaces, esto también es consecuencia de la Estrategia de Asignación de Recursos (Capítulo 5.1.2) incluida en el algoritmo GSTT para la búsqueda y obtención de soluciones.

En resumen, para la obtención de soluciones factibles de instancias medianas, las 5 metaheurísticas y GSTT no tienen mayor problema y todos mostraron efectividad en su desempeño.

Para tratar de obtener soluciones del UCTP con instancias grandes se ejecutaron en el laboratorio del CIICAp-UAEM los algoritmos ACO, GA, ILS, SA, TS y GSTT. Se efectuaron 20 pruebas para cada instancia grande (2 benchmarks) y con cada uno de los algoritmos ACO, GA, ILS, SA y TS para tratar de obtener los resultados que reporta en la literatura el grupo Metaheuristic Network [http:www.metaheuristic.org/]. Para probar el algoritmo GSTT se ejecutaron de él 100 pruebas con cada instancia grande (total 2x100=200 pruebas) y se sometieron a optimización 20 de las mejores soluciones.

MEJORES SOLUCIONES OPTIMAZADAS DE INSTANCIAS MEDIANAS

CON ALGORITMOS: ACO, GA, ILS, SA, TS Y GSTT

50 pruebas con cada instancia

VALORES DE FUNCIÓN OBJETIVO Y TIEMPO (Eficacia y Eficiencia)

Algoritmo	*Medium01*	*Medium02*	*Medium03*	*Medium04*	*Medium05*
ACO	182 en 887 seg	194 en 834 seg	252 en 916 seg	168 en 807 seg	176 en 897 seg
GA	181 en 315 seg	170 en 638 seg	246 en 567 seg	148 en 670 seg	208 en 758 seg
ILS	120 en 890 seg	121 en 699 seg	171 en 765 seg	107 en 745 seg	142 en 867 seg
SA	71 en 890 seg	70 en 724 seg	111 en 798 seg	65 en 801 seg	88 en 746 seg
TS	172 en 253 seg	183 en 713 seg	226 en 684 seg	163 en 508 seg	238 en 102 seg
GSTT	121 en 947 seg	111 en 1,098 seg	178 en 981 seg	91 en 1062 seg	120 en 676 seg

Pruebas efectuadas en Laboratorio de Cómputo del CIICAp-UAEM
en equipo: CPU Pentium 2.8 GHz, S.O. Linux

Cuadro 6.8: Mejores soluciones optimizadas de Instancias Medianas, obtenidas en laboratorio de CIICAp-UAEM

En general, de las pruebas efectuadas se pudo encontrar que la efectividad de los algoritmos ACO, GA, ILS, SA Y TS es congruente con la información reportada en la literatura. Unos obtienen algunas soluciones factibles y otros definitivamente no pueden obtener soluciones factibles. De las soluciones que son factibles, la eficacia y la eficiencia que obtienen los algoritmos es diferente para cada instancia grande, debido sobre todo a que, aunque ambas instancias están estructuradas con los mismos conjuntos de elementos (recursos), la complejidad computacional de cada instancia es diferente.

De los resultados obtenidos con las instancias grandes, ver Cuadro 6.9, en la columna dos se muestra la efectividad de las metaheurísticas a partir de la cantidad de soluciones factibles y sus porcentajes de eficiencia con respecto al número de pruebas realizadas. Para la instancia hard01, ACO, GA, SA 0 %, ILS 15 %, TS 10 %, de GAFD no hay dato, y GSTT 95 %; para la instancia hard02, GA y SA 0 %, ACO 5 %, ILS 80 %, TS 95 %, de GAFD no hay dato, y de GSTT se conserva el 95 %. Como se podrá observar, GSTT resulta obtener la mejor efectividad de todos; TS es uno de los algoritmos que aunque sus

soluciones no son muy eficaces siempre obtiene soluciones de las instancias grandes; en tercer lugar se encuentra ILS con porcentaje alto para el benchmark hard02, aunque un poco bajo para hard01 obtiene soluciones; ACO a veces obtiene soluciones, solo 5 % con hard02; GA y SA nunca obtienen soluciones; aunque de GA reportan en la literatura que obtiene 10 % con hard02, en CIICAp no se obtuvieron con ninguna instancia grande.

Continuando con los resultados mostrados en el Cuadro 6.9 para las instancias problema Hard01 y Hard02, en las columnas 3 y 4 se presentan las mejores soluciones sin optimizar y optimizadas respectivamente. De los resultados de las columnas 3 y 4 para GSTT, se puede ver que la Estrategia de Asignación de Recursos que se implementó como una de las aportaciones en este trabajo doctoral, funciona bastante bien, ya que el valor de la Función Objetivo (número de violaciones de instancias suaves) de 943 obtenido para Hard01 es mejor que el de ILS (1,057) y TS (1,372) que son los otros algoritmos que obtienen soluciones eficaces de esta instancia. La eficiencia también es mejor que los valores de ILS y TS. En la columna 4 que muestra las mejores soluciones optimizadas, el GSTT produce la mejor eficacia, ya que obtiene menos violaciones de restricciones suaves (52) mientras que los otros tres son mayores: ILS con 897, TS con 1,033 y GAFD con 1,212. Entonces, para la instancia Hard01 como entrada, la eficacia de GSTT resulta ser la mejor y con respecto al tiempo (eficiencia) consumido de proceso, también resulta ser mejor el GSTT. En las columnas 3 y 4 también, se muestran las mejores soluciones no optimizadas y optimizadas para la instancia hard02. Como se podrá ver en la columna 3, la mejor solución no optimizada la obtiene GSTT y también el mejor tiempo. Respecto a las soluciones optimizadas (columna 4), de los 5 algoritmos que pueden obtener soluciones factibles optimizadas, GSTT también resulta con mejor Función Objetivo (36 violaciones a restricciones suaves) muy lejos quedan ACO (717), ILS (759), TS (946) y GAFD (1,103). Por lo que una vez más, en relación a la eficacia resulta mejor el GSTT. Sobre la eficiencia con la instancia hard02 como entrada, GSTT obtiene 1,570.6 seg y resulta también en primer lugar, en segundo lugar esta TS con 1,921.12 seg. El resto de algoritmos (ACO e ILS) obtienen valores de tiempo (eficiencia) muy altos.

Los valores de Función Objetivo que entregan los algoritmos corresponden a las solu-

EFECTIVIDAD PARA OBTENER SOLUCIONES FACTIBLES Y
MEJORES SOLUCIONES DE INSTANCIA HARD01

20 PRUEBAS CON CADA ALGORITMO

Algoritmo	*Efectividad: No. soluciones factibles/ en 20 pruebas, % efectividad*	*Mejor solución Sin Optimización FO - seg (Eficacia-Eficiencia)*	*Mejor solución Optimizada FO - seg (Eficacia-Eficiencia)*
ACO	No obtuvo, 0%	No hay	No hay
GA	No obtuvo, 0%	No hay	No hay
ILS	3 / 20, 15%	1,057 - 6,101.3	897 - 8,993.86
SA	No obtuvo, 0%	No hay	No hay
TS	2 / 20, 10%	1,372 - 3,036.6	1,033 - 2,915.47
GAFD	No hay dato	No hay dato	1,212 - no da tiempo
GSTT	19 / 20, 95%	943 - 1,029.1	52 - 1,825.2

Pruebas efectuadas en Laboratorio de Cómputo del CIICAp-UAEM
en equipo: CPU Pentium 2.8 GHZ., S. O. Linux

GAFD: Genetic Algorithm with Forzed Diversity
GSTT: Generador de Soluciones de Timetabling.

EFECTIVIDAD PARA OBTENER SOLUCIONES FACTIBLES Y
MEJORES SOLUCIONES DE INSTANCIA HARD02

20 PRUEBAS CON CADA ALGORITMO

Algoritmo	*Efectividad: No. soluciones factibles/ en 20 pruebas, % efectividad*	*Mejor solución Sin Optimización FO - seg (Eficacia-Eficiencia)*	*Mejor solución Optimizada FO - seg (Eficacia-Eficiencia)*
ACO	1 / 20, 5%	1, 229 - 669.99	717 - 8,990.6
GA	No hubo, 0%	No obtuvo	No obtuvo
ILS	16 / 20, 80%	1,120 - 1,392.79	759 - 8,914.18
SA	No hubo, 0%	No obtuvo	No obtuvo
TS	19 / 20, 95%	1,563 - 261.29	946 - 1,921.12
GAFD	No hay dato	No hay dato	1,103 - no da tiempo
GSTT	19 / 20, 95%	945 - 115.5	36 - 1,570.6

Pruebas efectuadas en Laboratorio de Cómputo del CIICAp-UAEM, excepto GAFD,
en equipo: CPU Pentium 2.8 GHZ., S. O. Linux

GAFD: Genetic Algorithm with Forzed Diversity
GSTT: Generador de Soluciones de Timetabling.

Cuadro 6.9: Comparación de la efectividad para obtener soluciones factibles y mejores soluciones obtenidas de las instancias grandes Hard01 y Hard02

ciones factibles que logran obtener y que optimizan por medio de una búsqueda iterativa de mejores soluciones en el espacio de soluciones. La búsqueda de mejores soluciones en el espacio de búsqueda, puede en muchos casos extenderse por tiempos muy elevados, sin embargo la mejora de esas soluciones podría ser no sustancial. Entonces, para evitar cargar computacionalmente un proceso, al diseñar e implementar los algoritmos se ponen límites de búsqueda, considerando el número de ciclos o el tiempo para que en determinando momento se pare la búsqueda.

Capítulo 7

Conclusiones y Trabajo Futuro

En este trabajo doctoral, se hizo una investigación sobre el Problema de la Programación de Cursos en una Universidad, problema que corresponde al ámbito general de la programación de recursos en la optimización combinatoria. Para tratar este problema y obtener soluciones satisfactorias se diseño e implementó el algoritmo GSTT capaz de obtener soluciones factibles de instancias (benchmarks) consideradas grandes. Para la estructuración e implementación del algoritmo se aplicó la *Estrategia de Asignación de Recursos* que se explicó en el Capítulo 5.1.2, esta estrategia es una innovación propia que se puso en práctica con el fin de probar si ayudaría a obtener soluciones factibles de instancias problema de cualquier tamaño. Como resultado, la inclusión de esta estrategia, facilita programar la mayoría del número de eventos de la instancia, con lo cual se logra formar una solución parcial factible inicial. Posteriormente, los eventos restantes (una minoría) se programan a través de un reacomodo entre los horarios ya programados hasta completar la solución.

Con la puesta en práctica de la Estrategia de Asignación de Recursos en el diseño del GSTT, el algoritmo resultó ser muy eficiente debido al corto tiempo requerido para la obtención de una solución factible. También, la efectividad fue muy alta porque en promedio se tiene un $90\,\%$ de posibilidades de encontrar una solución factible en cada ejecución del GSTT. En conclusión, la Estrategia de Asignación de Recursos aplicada resultó un factor importante para que el GSTT logre obtener un porcentaje mayor de soluciones factibles que las que obtienen los algoritmos representativos encontrados en la literatura, los cua-

les utilizan otro tipo de estrategias. También se constató, que el GSTT tiene un excelente desempeño debido a que se diseñó con una estructura tridimensional que facilita la verificación de las restricciones para hacer la asignación de eventos. Es decir, una asignación de $clase = f(horario, dia, salon)$.

A partir de los resultados obtenidos en las pruebas realizadas, sobre todo con instancias grandes como fue el objetivo de este trabajo doctoral, se concluye que el algoritmo GSTT tiene muy buen desempeño, ya que obtiene soluciones factibles con mejor eficacia y eficiencia que los algoritmos con los que se comparó. Con instancias menores como las medianas, aunque no era el objetivo principal obtener soluciones de estas instancias, también obtiene muy buenas soluciones factibles y compite con los algoritmos representativos de la literatura. La eficacia y eficiencia con instancias grandes y medianas se demostró en todas las pruebas realizadas con las instancias problema tomadas de los benchmarks para problemas de Programación de Cursos en una Universidad. Respecto a la etapa de optimización, la metaheurística de *Recocido Simulado no resultó adecuada para optimizar las soluciones que se obtienen del UCTP*. Se comprobó que en el primer paso del análisis de sensibilidad, el recocido simulado optimiza las soluciones muy bien -a temperaturas altas-; pero estas soluciones ya no se mejoran en los siguientes tres pasos, ya que sólo logra optimizar 4 de 20 soluciones -efectividad del 20 %- con una de las dos instancias grandes y 2/20 soluciones -efectividad del 10 %- con la otra instancia grande. Por lo que se puede concluir, que el Recocido Simulado no es una buena metaheurística para la optimización de soluciones de problemas del tipo UCTP. En consecuencia, para optimizar las soluciones que obtiene el algoritmo GSTT resulta suficiente la búsqueda local iterada que se encuentra intrínseca en el Recocido Simulado. Por lo que una expectativa de mejora al algoritmo GSTT podría ser, quitar el Recocido Simulado para dejar fuera la variable de control de la temperatura y trabajar solo con Ciclos de Metrópolis para optimizar las soluciones.

Para cualquier caso particular en la calendarización de recursos, el algoritmo GSTT puede ser de gran utilidad como núcleo operativo para programar los horarios y espacios para desarrollar eventos; sin embargo, requerirá de modificaciones para adecuarlo a las

necesidades y los recursos disponibles en cada caso particular, ya que cada centro escolar, escuela superior, facultad o unidad académica tienen diferentes recursos, requisitos y necesidades, y que también son distintos en cada periodo escolar. Esto significa que el Modelo Matemático para cada caso particular puede diferir con respecto al que se presenta en este trabajo doctoral; sin embargo, el modelo aquí presentado servirá de base para obtener el caso particular de cualquier unidad académica.

Como trabajo futuro, se continuará dentro de la línea de investigación establecida en el cuerpo académico Optimización y Software de la UAEM, al cual se pertenece, y que es: la creación, mejora y aplicación de métodos y algoritmos en optimización combinatoria. A este respecto inclusive se trabaja también con algoritmos que no necesariamente funcionan con la aleatoriedad, por ejemplo con el método SIMPLEX que es para programación lineal, se han hecho trabajos de búsqueda de soluciones [Zavala-Diaz et al.,] para ponderar la problemática computacional de obtener soluciones con otras metodologías.

Además, existe el plan de hacer una aplicación particular del GSTT para la Facultad de Ciencias Químicas e Ingeniería (FCQeI) de la UAEM, ya que esta facultad tiene una población muy alta de estudiantes y profesores y gran dificultad para programar sus horarios escolares. Por esta problemática, la FCQeI esta muy interesada en que se haga una implementación del GSTT para ellos y en donde se incluyan a los profesores como un recurso más, ya que cuentan con diferentes tipos de profesores, lo que incrementará más la complejidad del problema. En este proyecto se involucrarán algunas tesis de maestría y licenciatura para realizar la implementación. Actualmente, ya se tiene parte del proyecto implementado donde se involucraron dos tesis de licenciatura. De estos trabajos, se tiene la conformación de una base de datos de los profesores y parte de la interfaz para dar de alta de forma automatizada los posibles horarios en que un profesor daría sus clases. Para la implementación particular en la FCQeI, se pretende incluir como metaheurística a ILS en la parte de optimización. Por lo tanto, se tendría que adecuar el algoritmo GSTT para obtener el modelo matemático apropiado para la FCQeI y hacer las modificaciones necesarias para incluir la metaheurística ILS. Para este propósito, se partiría del Modelo Matemático aquí desarrollado y se anexarían las restricciones duras (ejemplo:

los profesores) y restricciones suaves (ejemplo: los profesores titulares). Además de las necesidades o requisitos particulares de la FCQeI, por ejemplo: los salones.

También se trabajará en mejorar las cualidades de eficacia y eficiencia del propio algoritmo GSTT. De estas mejoras, se pretende hacer modificaciones a la estructura tridimensional y a la estrategia de asignación de recursos. Respecto a la optimización se contemplan dos posibilidades mediatas, la primera: excluir el recocido simulado e integrar otra metaheurística, la segunda: hacer una combinación del recocido simulado, ya integrado en el GSTT, con otra metaheurística. Con este último fin ya se esta trabajando con un algoritmo híbrido para búsqueda local que esta compuesto de el SA con ligeras modificaciones y se integró en un Algoritmo Genético [Cruz-Chávez et al., 2010a]. También, se esta tratando de aprovechar las posibilidades de comunicación entre dos clusteres (Morelos-Veracruz): la MiniGrid Tarántula instalada en el Tecnológico de Veracruz y la Grid instalada por el cuerpo académico Optimización y software en el CIICAp. El fin de esta posibilidad es mostrar que el híbrido con SA y un algoritmo genético es más eficiente al trabajar en ambiente Grid [Cruz-Chávez et al., 2010b] al aprovechar la exploración y explotación del espacio de soluciones. Trabajar en estas posibilidades implica extender la investigación realizada en las posibilidades propuestas para obtener mejores algoritmos y tratar de obtener soluciones más cercanas a la óptima o quizá alcanzar la óptima con tiempos de computación reducidos.

Bibliografía

[Aarts and Korst, 1989] Aarts, E. and Korst, J. (1989). *Simulated Annealing and Boltzmann Machines: a stochastic approach to combinatorial optimization and neural computing*. John Wiley & Sons, Great Britain. pages 272.

[Abramson, 2001] Abramson, D. (2001). Constructing school timetables using simulated annealing: sequential and parallel algorithms. *Management Science, Vol. 37, No. 1*, pages 98–113.

[Abramson and Abela, 1992] Abramson, D. and Abela, J. (1992). A parallel genetic algorithm for solving the school timetabling problem. Technical report, Division of information technology, C.S.I.R.O. University of Edinburg.

[Appleby et al., 1960] Appleby, J. S., Blake, D. V., and Newman, E. A. (1960). Techniques for reducing school timetables on a computer and their application to other scheduling problems. *The Computer Journal, Vol. 3*, pages 237–245.

[Baeck et al., 2000] Baeck, T., Fogel, D., and Michalewicz, Z. (2000). Evolutionary computation 1: Basic algorithms and operators. *Institute of Physics*.

[Blum and Roli, 2003] Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys, Vol. 35, No. 3*, pages 268–308.

[Brelaz, 1979] Brelaz, D. (1979). New methods to color the vertices of a graph. *Communications of ACM, Vol. 22*, pages 251–256.

[Burke and Carter, 1997] Burke, E. and Carter, M., editors (1997). *The Practice and Theory of Automated Timetabling: Selected Papers from the Second International Conference*, Lectures Notes in Computer Science 1408. Springer-Verlag, Berlin Heidelberg.

[Burke et al., 1994] Burke, E., Elliman, D., and Weare, R. (1994). A university timetabling system based on graph colouring and constraint manipulation. *Journal of research on computing in education, Vol. 27 Iss. 1*, pages 1–18.

[Burke et al., 2004] Burke, E., Kendall, G., and Soubeiga, E. (2004). A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristic, Vol. 9. Kluwer Academic Publishers, Manufactured in the Netherlands*, pages 451–470.

[Carter et al., 1994] Carter, M. W., Laporte, G., and Chinneck, J. W. (1994). A general examination scheduling system. *Interfaces, Vol. 24*, pages 109–120.

[Cerny, 1985] Cerny, V. (1985). An thermodynamical approach to the traveling salesman problem. *Journal of Optimization Theory and Applications, Vol. 45*, pages 41–51.

[Cheng et al., 2003] Cheng, E., Kruk, S., and Lipman, M. (2003). On the multicommodity flow formulations for the student scheduling problem. *Congressus Numeratium, Vol. 160*, pages 177–181.

[Chiarandini and Stützle, 2002] Chiarandini, M. and Stützle, T. (2002). Experimental evaluation of course timetabling algorithms. Technical report, FG Intellektik. TU Darmstadt, Germany.

[Colorni et al., 1998] Colorni, A., Dorigo, M., and Maniezzo, V. (1998). Metaheuristics for high-school timetabling. *Computational Optimization and Applications Journal Vol. 3 No. 9*, pages 277–298.

[Cook, 1971] Cook, S. (1971). The complexity of theorem proving procedures. In *Proc. of 3º Annual ACM Symposium on the Theory of Computing*, pages 151–158.

[Cook, 2000] Cook, S. (2000). P versus NP problem. *Manuscript prepared for the clay Mathematics Institute for the Millenium Prize Problem*. April.

[Cruz-Chávez, 2004] Cruz-Chávez, M. A. (2004). *Cooperación de Procesos para el Problema de Job Shop Scheduling Aplicando Recocido Simulado*. PhD thesis, Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Morelos.

[Cruz-Chávez et al., 2006] Cruz-Chávez, M. A., Frausto Solís, J., Zavala-Díaz, J. C., Sanvicente-Sánchez, H., and Cruz-Rosales, M. H. (2006). A simulated annealing algorithm with cooperative processes for scheduling problems. *Lecture Notes in Computer Science, Springer Verlag Pub., Berlin Heidelberg, ISSN: 0302-9743, Vol.*, page to appear.

[Cruz-Chávez et al., 2010a] Cruz-Chávez, M. A., Rodríguez-León, A., Ávila Melgar, E. Y., Juárez-Pérez, F., and Cruz-Rosales, M. H. (2010a). Genetic-annealing algorithm in grid environment for scheduling problems, communication in computer and information science: Security-enriched urban computing and smart grid. *Springer Verlag Pub. Berlin Heidelberg, ISSN: 1865-0929, Vol. 78.*, pages 1–9.

[Cruz-Chávez et al., 2010b] Cruz-Chávez, M. A., Rodríguez-León, A., Ávila Melgar, E. Y., Juárez-Pérez, F., and Cruz-Rosales, M. H. (2010b). Gridification of genetic algorithm with reduced communication for the job shop scheduling problem. *International Journal of Grid and Distributed Computing Science and Engineering Research Support Society, Australia, ISSN: 2005-4262, Vol. 3, No. 3*, pages 13–28.

[Cruz-Rosales et al., 2010] Cruz-Rosales, M. H., Cruz-Chávez, M. A., and Rivera, A. D. (2010). Mathematical model and a heuristic solution proposal with a generator of instances for the university course timetabling problem. *Journal of Scheduling*.

[de Werra, 1985] de Werra, D. (1985). An introduction to timetabling. *European Journal of Operational Research, Vol. 19*, pages 151–162.

[de Werra et al., 1985] de Werra, D., Pasche, C., and Petter, A. (1985). Time-tabling problems: should they be canonical? *INFOR, Vol. 14, No. 4, November*, pages 304–308.

[Dorigo et al., 1996] Dorigo, M., Maniezzo, V., and Colorni, A. (1996). The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Vol. 26*, pages 29–41.

[Dowsland and Adenso Díaz, 2001] Dowsland, K. A. and Adenso Díaz, B. (2001). Diseño de heurísticas y fundamentos del recocido simulado. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial No. 20. ISSN: 1137-3601*, pages 34–52.

[Elmohamed and Fox, 1997] Elmohamed, S. and Fox, G. (1997). A comparison of annealing techniques for academic course scheduling. *Selected papers from the Second International Conference on Practice and Theory of Automated Timetabling II. Ed. Edmund Burke and Mike Carter, Lecture Notes in Computer Science, Springer*, pages 92–114.

[Faber et al., 1998] Faber, W., Leone, N., and Pfeifer, G. (1998). Representing school timetabling in a disjunctive logic programming language. In *Proceedings of the 13th Workshop on logic programming (WLP'98)*.

[Glover and Laguna, 1998] Glover, F. and Laguna, M. (1998). Tabu search. *Kluwer Academic Publishers, Boston et Al.*

[Hertz, 1992] Hertz, A. (1992). Tabu search for large scale timetabling problems. *European Journal of Operational Research, Vol. 54*, pages 39–47.

[Hertz and Widmer, 2003] Hertz, A. and Widmer, M. (2003). Guidelines for the use of meta-heuristics in combinatorial optimization. *European Journal of Operational, Vol. 151*, pages 247–252.

[http://iridia.ulb.ac.be/Supp/IridiaSupp2002-001/index.html, 2009] http://iridia.ulb.ac.be/Supp/IridiaSupp2002-001/index.html (febrero 2009).

[http://www.metaheuristic.org/, 2009] http://www.metaheuristic.org/ (febrero 2009).

[Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, Number 4598, Vol. 220*, pages 671–680.

[Kostuch, 2005] Kostuch, P. (2005). The university course timetabling problem with a three-phase approach. *Burke, E. and Trick M. (Eds.): PATAT 2004, LNCS 3616, Springer-Verlag Berlin Heidelberg*, pages 109–125.

[Laarhoven and Aarts, 1992] Laarhoven, V. P. and Aarts, E. (1992). Simulated annealing: Theory and application. *D. Reidel Publishing Company, Dordrecht, Holland, Netherlands. ISBN: 90-277-2513-6*, page 187.

[Lourenço et al., 2001] Lourenço, H., Martin, O., and Stützle, T. (2001). A beginner's introduction to iterated local search. In *Proceedings 4th. Metaheuristics International Conference*, pages 1–6.

[Lourenço et al., 2002] Lourenço, H., Martin, O., and Stützle, T. (2002). Iterated local search. *Glover, F. and Kochenberger, G. (eds.), Handbook of Metaheuristics, Vol. 57 of International Series in Operations Research & Management. Kluwer Academic Publishers*, pages 321–353.

[Martin and Otto, 1996] Martin, O. and Otto, S. (1996). Combining simulated annealing with local search heuristics. *Annals of Operating Research, Vol. 63*, pages 57–75.

[Martí, 2003] Martí, R. (2003). Procedimientos metaheurísticos en optimización combinatoria. *Matemátiques Vol. 1, No. 1*, pages 3–62.

[Melício et al., 2003] Melício, F., Caldeira, J., and Rosa, A. (2003). Two neighbourhood approaches to the timetabling problem.

[Metrópolis et al., 1953] Metrópolis, N., Rosenbluth, A. W., Rosenbluth, M.Ñ., Teller, A. H., and Teller, E. (1953). Equation of state calculation by fast computing machines. *Journal of Chemistry Physics, Vol. 21*, pages 1087–1091.

[Paechter et al., 1998] Paechter, B., Rankin, R. C., Cumming, A., and Fogarty, T. C. (1998). Timetabling the classes of an entire university with and evolutionary algorithm. *Parallel Problem Solving from Nature (PPSN) V. Lectures Notes in Computer Science 1498, Springer-Verlag, Berlín*, pages 865–874.

[Papadimitriou, 1994] Papadimitriou, C. (1994). *Computational complexity*. Addison Wesley Pub. Co., USA. ISBN 0-201-53082-1, pages 523.

[Papadimitriou and Steiglitz, 1982] Papadimitriou, C. and Steiglitz, K. (1982). *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, Englewood Cliffs, New Jersey, USA. ISBN 0-13-152462-3, pages 496.

[Papadimitriou and Steiglitz, 1998] Papadimitriou, C. and Steiglitz, K. (1998). *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications Inc. New York, USA.

[Paquete and Stützle, 2002] Paquete, L. and Stützle, T. (2002). Experimental investigation of iterated local search for coloring graphs. In *S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf and G. Raidl (eds.), Applications of Evolutionary Computing, Proceedings of EvoWork-shops 2002. Lectures Notes in Computer Science 2279, Springer-Verlag, Berlin*, pages 122–131.

[Pinedo, 2002] Pinedo, M. (2002). *Scheduling: Theory, Algorithms and Systems*. 2nd Ed. Prentice Hall, New York, USA.

[Reeves, 1999] Reeves, C. (1999). Landscapes, operators and heuristics search. *Annals of Operations Research, Vol. 86*, pages 473–490.

[Ross et al., 1994] Ross, P., Corne, D., and Fang, H. (1994). Improving evolutionary timetabling with delta evaluation and directed mutation. *H. P. Schwefel and Y. Davidor and R. Manner (eds). Parallel Problem Solving from Nature (PPSN) III. Lectures Notes in Computer Science, Springer-Verlag*, pages 560–565.

[Rossi-Doria et al., 2002] Rossi-Doria, O., Blum, C., Knowles, J., Sampels, M., Socha, K., and Paechter, B. (2002). A local search for the timetabling problem. Technical report, TR/IRIDIA/2002-16 de julio. Presentado en PATAT2002. http://www.metaheuristics.org/.

[Rossi-Doria et al., 2003] Rossi-Doria, O., Sampels, M., Chiarandini, M., Knowles, J., Manfrin, M., Mastrolilli, M., Paquete, L., and Paechter, B. (2003). A comparison of the performance of different metaheuristics on the timetabling problem. *Burke, E. and De Causmaecker, P. (eds.): Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin*, pages 329–351.

[Sanvicente-Sanchez and Frausto, 2004] Sanvicente-Sanchez, H. and Frausto, J. (2004). Method to establish the cooling scheme in simulated annealing like algorithms. In *International Conference ICCSA'2004. LNCS Vol. 3095*, pages 755–763.

[Schaerf, 1999] Schaerf, A. (1999). A survey of automated timetabling. *Artif. Intell. Rev. 13*, pages 87–127.

[Smith, 1975] Smith, G. (1975). On maintenance of the opportunity list for class-teacher timetable problems. *Communications of the ACM, Vol. 18, No. 4*, pages 203–208.

[Thompson and Dowsland, 1996] Thompson, j. and Dowsland, K. (1996). Variants of simulated annealing for the examination timetabling problem. *Annals of Operations Research, Vol. 63*, pages 105–128.

[Zavala-Diaz et al.,] Zavala-Diaz, J. C., Cruz-Chávez, M. A., and Cruz-Rosales, M. H. Mathematical multi-objective model for the selection of a portfolio of investment in the mexican stock market.

[Zervoudakis and Stamatopoulos, 2001] Zervoudakis, K. and Stamatopoulos, P. (2001). A generic object-oriented constraint-based model for university course timetabling. *E. Burke and W. Erben (Eds.): PATAT 2000, LNCS 2079, Springer-Verlag Berlin Heidelberg*, pages 28–47.