Universidad Tecnológica de la Mixteca

"SOLUCIONES APROXIMADAS PARA EL PROBLEMA DE HORARIOS DE LA UTM A TRAVÉS DE UNA HIPER-HEURÍSTICA BASADA EN BÚSQUEDA TABÚ"

Tesis

para obtener el grado de

maestro en tecnologías de cómputo aplicado

presenta:

Juan Pablo Gómez Martinez

Director de tesis:
Dra. Lluvia Carolina Morales Reynaga
Co-Director de tesis:
Dr. José Figueroa Martínez

huajuapan de león, oaxaca, mayo de 2018

A mi familia, en especial a mis padres y a mi hija Luz Marianyt.

Agradecimientos

A mi directora de Tesis Dra. Lluvia Carolina Morales Reynaga, por su paciencia, ayuda, confianza, guía y su tiempo.

A mi co-director de Tesis Dr. José, por el apoyo que me proporcionó durante la elaboración del presente trabajo.

A mis sinodales Dr. Ricardo Pérez Águila, Dr. Felipe Trujillo Romero, Mtra. Verónica Rodríguez López, Dr. Manuel Hernández Gutiérrez, Dr. Agustin Santiago.

A la Universidad Tecnológica de la Mixteca. A los profesores y compañeros que me acompañaron en mi trayecto, por las buenas experiencias compartidas y el apoyo durante la estadía en la UTM.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

La secuenciación y calendarización es una forma de toma de desiciones que juega un papel crucial en las industrias de manufactura y de servicios. En el ambiente actual de competencia, hacer una secuenciación y calendarización se ha vuelto una necesidad para sobrevivir en el mercado [41]. La calendarización siempre ha tomado parte importante en las actividades del hombre, desde tomar un transporte, asistir a un evento o cita, ir a trabajar, presentar un examen, etc.; así, sin una asignación de tiempo adecuado a cada tarea, éstas no se pueden realizar en el tiempo requerido, lo que puede provocar problemas económicos en la industria[43] (retraso en la entrega de algún producto, retraso en algún pago), de salud (retraso en cirugías, ingesta de medicamentos antes o después de tiempo, etc.)[41] y otros más.

Este tipo de problemas son generalmente difíciles de resolver en forma exacta, ya que son problemas de optimización y crecen de manera exponencial según el tamaño del problema, de manera que es inviable la aplicación de los mismos dado la gran cantidad de recursos necesarios para solucionarlos. En este punto aparecen los métodos heurísticos como una alternativa para solucionar estos problemas[12]. Aunque estos métodos no pueden garantizar soluciones óptimas o en casos extremos pueden no llegar a encontrar soluciones cercanas al óptimo, sin embargo, encuentran soluciones aceptables con un consumo bajo de recursos y en un tiempo moderado. En general las heurísticas no recorren todo el espacio de soluciones, obteniendo solo soluciones localmente óptimas[28].

En un nivel superior de abstracción a las heurísticas están las Meta-heurísticas que son una clase de métodos aproximados que están diseñados para resolver problemas de difícil optimización combinatoria, en los que los heurísticos clásicos no son efectivos.Los Meta-heurísticos proporcionan un marco general para crear nuevos algoritmos híbridos, combinando diferentes conceptos de diversos campos como la genética, la biología, la inteligencia artificial, las matemáticas, la física y la neurología, entre otras. Actualmente existen varias investigaciones con respecto al problema de calendarización universitario por lo que se puede encontrar una gran variedad de bibliografía respecto al tema, entre ellas [10] y [40] que abordan el problema de CT (Calendarización de Cursos o *Course Timetabling* por sus siglas en inglés) y UCTTP (Problema de calendarización de cursos de una Universidad o *University Course Timetabling* por sus siglas en inglés) comparando y analizando varios enfoques tales como los métodos de investigación operativa, métodos heurísticos, enfoques multi-objetivo y métodos inteligentes novedosos para resolver el problema de UCTTP; también en [49] y [51] utilizan meta-heurísticas como algoritmos evolutivos y el algoritmo de coloración de grafos para resolver el problema de CT. Resolver este tipo de problemas de manera manual hace que las Universidades pierdan demasiado tiempo tratando de hacer un buen horario, normalmente semanas o meses.

En esta tesis se propone un algoritmo hiper-heurístico para resolver el problema de horarios de la Universidad Tecnológica de la Mixteca (UTM), utilizando cuatro algoritmos meta-heurísticos para este propósito: Algoritmo Genético, Algoritmo de Recocido Simulado, Algoritmo de Búsqueda Tabú y Algoritmo de Búsqueda Armónica.

1.1. Planteamiento del Problema

Tanto las meta-heurísticas como la hiper-heurística son métodos que buscan soluciones buenas (cercanas al óptimo) en un tiempo razonable, ya que es imposible encontrar una solución exacta en un tiempo corto, tomando en cuenta que la UTM es una Universidad que cuenta con aproximadamente 2000 estudiantes, 200 profesores, 298 espacios y 246 materias (en el ciclo escolar 2017-B); éstos datos hacen que el problema de horarios de la UTM sea difícil de resolver de manera exacta.

Actualmente, en la UTM se le dedica bastante tiempo (tres semanas en promedio) a la elaboración de los horarios de clases de cada semestre, ésto se debe al hecho de que se tienen que satisfacer varias restricciones fuertes(*hard constraints*) y restricciones débiles(*soft constraints*) a la hora de generar los horarios. En el Capítulo 2 se aborda más detalladamente el tema de las restricciones enfocándose en las restricciones de la UTM.

Las restricciones fuertes representan por sí mismas un problema bastante complejo cuando se quieren satisfacer de manera manual, ya que se pueden solapar las horas en algún momento y es tedioso tener que reasignar las horas sin que en otro punto suceda lo mismo. Es posible que cuando se automatice el sistema de horarios, se puedan satisfacer la mayor parte de las restricciones débiles ya que el sistema se desarrollará de tal manera que la meta-heurística o hiper-heurística sea la que cumpla con todas las restricciones fuertes y, de ser posible, la mayor parte de las restricciones débiles, es decir, que se minimice la función heurística. Este problema, en su representación mínima, es un típico problema de calendarización. Con la representación adecuada, podría resolverse a través de una hiper-heurística en mucho menos tiempo y con la posibilidad de proporcionar posibles soluciones a los encargados de elaborar los horarios, además de tener en cuenta horarios y condiciones mucho más complejas y opuestas, en ciertos casos.

1.2. Justificación

Una de las razones que motiva el desarrollo de este proyecto es el tiempo que se puede ahorrar al momento de generar un horario de clases de tal manera que la solución sea lo más aproximado a una solución exacta y que además dicha solución sea de utilidad para el encargado de generar los horarios en la UTM. Con el desarrollo de una hiper-heurística que permita la generación automática de horarios de clases factibles, dadas las condiciones mencionadas en la sección anterior, representadas por dos funciones que deberán ser minimizadas (función de factibilidad y función heurística), la UTM se verá beneficiada al satisfacer la mayor parte de las necesidades que los profesores tienen para dar sus clases, obteniendo así diversas propuestas de horarios de clases en un tiempo mucho menor al que se tendría al tratar de resolver el problema de manera manual.

Esta hiper-heurística se puede utilizar en otras instituciones del SUNEO (Sistema de Universidades Estatales de Oaxaca) a través del sistema de horarios desarrollado para la UTM sin la necesidad de realizar muchos cambios en dicho sistema lo que implica cambios mínimos también en la representación del problema manipulada por la hiper-heurística. Esto debido a que la mayoría de las condiciones débiles contempladas están basadas en el modelo del SUNEO[45].

Dicho modelo se basa en una jornada de tiempo completo tanto para profesores como para estudiantes, en la que los estudiantes pueden y deben tener clases de 7 a 3 y/o de 16 a 19 horas y los profesores tienen que estar presentes en la universidad al menos 8 horas dentro del horario anterior, dependiendo de su horario asignado. Por otro lado, dicho modelo también contempla actividades de esparcimiento, deportivas, de fomento a la cultura, divulgación, gestión, investigación, gestión, lectura individual en la biblioteca o en círculos de lectura, sistema de tutorías, horarios de asesorías, clases de inglés, entre otras actividades que promueven que los miembros de la comunidad universitaria se encuentren de manera presencial en esta casa de estudios tanto por la mañana como por la tarde.

Como el sistema por medio del cual se obtiene la información, se mostrarán los horarios al encargado final de los mismos, se trata de un sistema de recomendación; además la hiper-heurística deberá permitir generar varios resultados para que finalmente el administrador del sistema pueda elegir la solución que le parezca más adecuada, todo esto sin el esfuerzo adicional que tendría un ser humano al realizar la misma tarea, es decir, en cuestión de horas o días y no de semanas como se mencionaba anteriormente.

Por otro lado, esta tesis reactivará la investigación de la UTM en ésta área, misma que está siendo apoyada por esta institución con la disposición de información de diferentes semestres y periodos escolares, así como la captura de la misma.

1.3. Hipótesis

Es posible diseñar un algoritmo hiper-heurístico que permita recomendar una solución inicial lo más factible posible de calendarización de horarios de clase de la Universidad Tecnológica de la Mixteca.

1.4. Trabajos Relacionados

En esta sección se recopila información a través de literatura referentes a la resolución del problema de horarios a través de meta-heurísticas e hiper-heurísticas y al problema de calendarización en general, para así tener una base adecuada que permita utilizar las mejores herramientas en el desarrollo de la tesis en el aspecto técnico y en la parte del diseño y representación de la solución al problema de horarios de la UTM.

En [10] los autores hacen un estudio sobre los enfoques para el calendarización de una universidad basándose en diferentes conjuntos de datos y se obtiene el desempeño de cada algoritmo al mandarle los datos de entrada con diferentes tamaños. En este artículo se utilizan diferentes enfoques de algoritmos heurísticos para resolver el problema, por mencionar los métodos de Investigación Operativa, métodos heurísticos, nuevos métodos inteligentes y enfoque basado en sistemas multi-agentes distribuidos. Al final se explica que los enfoques basados en métodos de investigación operativa no tienen buena eficiencia resolviendo este tipo de problemas, sin embargo son fáciles de implementar; por otro lado, la exploración del espacio de soluciones tiene un desempeño más eficiente aplicando métodos meta-heurísticos y nuevos métodos inteligentes tales como métodos híbridos y enfoques difusos para la solución de este tipo de problemas.

En este artículo se llega a la conclusión de que las meta-heurísticas tienen buena eficiencia a la hora de resolver UCTTP, por lo que en el problema de la UTM se pueden aplicar meta-heurísticas y dejar de lado los otros enfoques, no obstante no se puede asegurar que para el problema de calendarización de la UTM no existan mejores enfoques para resolver éste problema.

En [47] se utiliza el enfoque ILS (Búsqueda Local con Iteración o *Iterative Local Search* por sus siglas en inglés), utilizando una hiper-heurística el cual genera heurísticas basados en un número fijo de operaciones de agregación y eliminación (utilizando una lista). Para las pruebas de desempeño se utilizan dos problemas diferentes, usando puntos de referencia del problema de horarios del mundo real citados de la segunda competición Internacional de calendarización. El resultado muestra que mezclando operaciones de

agregación y eliminación con un framework ILS genera un enfoque efectivo de hiper-heurística.

Se puede aplicar el enfoque ILS al problema de calendarización de la UTM, y aunque en los resultados del artículo se ve que este enfoque no mejora mucho los resultados en comparación con aplicar solo una metaheurística al problema de calendarización, en unos casos si lo hace, por lo tanto se puede ver como una posible opción para resolver el UCTTP de la UTM.

En [40] se citan algunos algoritmos para resolver el STP (Problema de calendarización de una escuela o *School Timetabling Problem* por sus siglas en inglés), por mencionar el algoritmo de abejas, programación de restricciones, métodos de satisfacción de restricciones, transferencias cíclicas, algoritmos evolutivos, etc. También se habla de los tipos de formatos de los datasets para el STP y al final aborda el tema de algunos paquetes de software que se han desarrollado en diferentes países para resolver el problema.

En los paquetes de software que menciona este artículo se puede observar que México no se encuentra en la lista de autores de los mismos y, aunque haya software para universidades, las restricciones deseables no se pueden satisfacer por completo porque aún no se cuenta con un generador de horarios "universal".

En [49] se presenta la herramienta de EAT (Calendarización basado en Algoritmos Evolutivos o *Evolutionary Algorithms based timetabling* por sus siglas en inglés) para resolver el problema de CTP (Problema de calendarización de cursos o *Course Timetabling Problem* por sus siglas en inglés), los algoritmos utilizados por esta herramienta son el algoritmo genético y el algoritmo memético, ambos basados en población y búsqueda múltiple direccional. Se hace una comparativa de 14 ejecuciones para mostrar el desempeño de los algoritmos modificados, se identifica qué factores e interacciones fueron estadísticamente significativos, también se identifican los parámetros apropiados para el GA (*Genetic Algorithm* por sus siglas en inglés o Algoritmos Genéticos en español) y el MA (*Memetic Algorithm* por sus siglas en inglés o Algoritmo Memético en español) y al final se compara el desempeño de varios algoritmos híbridos. El algoritmo genético

modificado con operadores del algoritmo memético aumentó su desempeño en un cincuenta por ciento.

Este artículo describe a fondo la manera en que se puede aplicar un algoritmo genético a un UCTTP, ya que la herramienta desarrollada en el artículo se enfoca a las Universidades, proporcionando así una idea de los parámetros iniciales que se pueden asignar a la meta-heurística para poder obtener mejores resultados.

En [51] se describe el diseño e implementación de un sistema de calendarización de cursos con doble objetivo para la Facultad de Ciencias del colegio de Rollins en Estados Unidos, se hace una comparación de los resultados del sistema con otro sistema del mismo colegio construido de manera manual. El sistema tiene una interfaz gráfica (GUI) que habilita la participación del usuario en la entrada, construcción y modificación de un horario. El sistema es capaz de permitirle al usuario asignar o reasignar directamente los cursos a los tiempos mientras son guiados por las heurísticas. El objetivo primario en este sistema es minimizar los conflictos, y el objetivo secundario de proximidad es crear horarios compactos para los instructores y estudiantes. El sistema utiliza un sistema de coloración de grafos para representar el problema.

El problema de calendarización que abordan en este artículo es principalmente para el colegio de Rollins, sin embargo se pueden rescatar datos de la interfaz gráfica tales como su estructuración o los conflictos que se hayan presentado durante su desarrollo.

En el año 2000 se desarrolló un prototipo de sistema de horarios para la UTM utilizando algoritmos genéticos [35], sin embargo en ese entonces no se contaba con el número que se tiene actualmente de aulas, profesores, alumnos y carreras, además en dicho proyecto no se plantearon las restricciones que se van a contemplar en el algoritmo hiper-heurístico a proponer ni con la comparativa con diferentes meta-heurísticas para seleccionar a la mejor meta-heurística base.

Actualmente, se cuenta con un sistema que facilita la obtención de los datos tanto de profesores, alumnos, materias y aulas que ya están ingresados en las bases de datos de la UTM. Éste sistema requiere un algoritmo inteligente para que genere los horarios, de modo que se puedan adaptar a

los requerimientos de los profesores y/o alumnos, definidos como parte de las restricciones débiles que se mencionarán en la siguiente sección.

Como podemos ver en los párrafos anteriores el problema de UCTTP es bastante complejo y habrá que determinar una forma eficiente de representarlo para que se pueda resolver a través de una hiper-heurística. Los autores citados abordan el problema de calendarización utilizando diferentes enfoques meta-heurísticos, sin embargo no se encontraron muchos artículos acerca de resolver el problema utilizando algún enfoque hiper-heurístico enfocado al problema de calendarización Universitario.

1.5. Objetivos

1.5.1. Objetivo General

Diseñar e integrar un algoritmo hiper-heurístico a partir de la comparación del desempeño de cuatro algoritmos meta-heurísticos para brindar recomendaciones de soluciones iniciales aproximadas a los responsables del proceso de calendarización de horarios de clase de la Universidad Tecnológica de la Mixteca.

1.5.2. Objetivos Específicos

- Investigar y documentar el estado del arte del problema de calendarización de cursos, así como de meta-heurísticas que lo resuelven.
- Representar el problema de horarios de la Universidad Tecnológica de la Mixteca de manera que pueda ser utilizado y evaluado por diferentes meta-heurísticas.
- Probar e identificar las mejores meta-heurísticas que integrarán una hiper-heurística para la solución del problema de calendarización de cursos de la Universidad Tecnológica de la Mixteca.

1.6. Metas

- Contar con el marco teórico necesario para abordar el problema de calendarización y su representación.

- Obtener una representación del problema de la Universidad Tecnológica de la Mixteca que permita lidiar con las restricciones fuertes del mismo.
- Implementar los algoritmos meta-heurísticos elegidos para resolver el problema de calendarización.
- Proponer un algoritmo hiper-heurístico en base a las pruebas preliminares llevadas a cabo con las meta-heurísticas implementadas en el paso anterior.
- Implementar el algoritmo hiper-heurístico propuesto y hacer las pruebas correspondientes.
- Obtener un reporte de pruebas de eficiencia de los algoritmos utilizados, analizando su eficiencia y bondad en términos de tiempo y función de factibilidad/heurística.

1.7. Metodología

En la Figura 1.1 se presenta la metodología a seguir durante el desarrollo del proyecto de tesis para alcanzar el objetivo propuesto. Para lograr este objetivo, es necesario abordar el tema de calendarización tomando en consideración los aspectos principales (variables) de la UTM y así poder obtener un modelo que represente la función objetivo de tal manera que ésta función pueda ser utilizada en los algoritmos meta-heurísticos generando resultados factibles.

Ya que el problema se trata de un problema del mundo real (en términos de aplicabilidad), para su resolución se plantean tres etapas [26]:

1.- Primero se presenta la etapa de formulación que abarca la tanto la obtención de información, así como la representación del problema, en esta etapa se definirá la información que va a ser incluida en el modelo a partir de las características principales de la universidad y el modelo del SUNEO, generando así las restricciones fuertes y débiles. Cabe mencionar que hay que abstraer lo más que se pueda el problema para minimizar la complejidad y obtener resultados factibles en un tiempo razonable.

2.- Después de la formulación se presenta la etapa de modelación y solución en donde la definición del problema se plantea en un lenguaje que el programa o la meta-heurística pueda utilizar. En este punto se plantean la función heurística y de factibilidad, las cuales se presentan como un modelo matemático para facilitar la búsqueda de la solución.

3.- En los capítulos posteriores se presentan la implementación de los algoritmos propuestos con sus respectivas pruebas y comparaciones, llegando así al objetivo final donde el mejor algoritmo; tomando en cuenta características como tiempo de ejecución y bondad de la solución; se integre al sistema actual de horarios de la UTM.

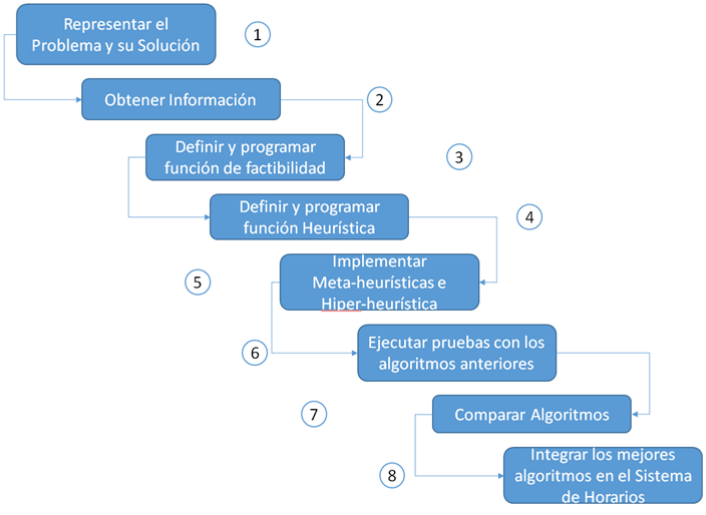


Figura 1.1: Metodología para desarrollar el algoritmo del sistema de horarios

Capítulo 2

Marco Teórico

En este capítulo se definen los elementos, técnicas o clasificaciones más utilizadas en los problemas de horarios, se debe definir cada elemento ya que la mayor parte de ellas se utilizan durante el desarrollo de la tesis y es necesario para un correcto entendimiento sobre el tema.

2.1. El problema de Calendarización de Horarios

En esta sección se definen: el problema de calendarización, los tipos de horarios, las restricciones y la complejidad; todo lo anterior para introducir el concepto de tal manera que se pueda entender el porqué es importante atacar el problema de horarios con algoritmos meta-heurísticos.

Los horarios son estructuras organizacionales que pueden ser encontrados en diferentes áreas de las actividades humanas incluyendo deportes, entretenimiento, transporte, industria y educación[14]. Al proceso de seleccionar y asignar recursos y tiempo al conjunto de actividades de una planificación, y teniendo en cuenta que la asignación debe cumplir con un conjunto de restricciones que reflejan la relación temporal entre las actividades y la capacidad limitada de los recursos compartidos se le llama Problema de Programación o Problema de Calendarización [21].

El *Problema de Programación de Horarios* es una forma particular del Problema de Calendarización. De acuerdo a [38] la programación de horarios se divide en tres categorías con características similares:

- *Programación de horarios para evaluaciones* o *Examination Timetabling Problem* (ETP): Se caracteriza por asignar un conjunto de exámenes a un número limitado de periodos de tiempo sujetos a un conjunto de restricciones. Estas restricciones usualmente se clasifican en restricciones fuertes y débiles [9].

- *Programación de horarios de clases para escuelas* o *School Timetabling Problem* (STP): Es un tipo de asignación de horarios para salones o cursos, está definido en términos del número de grados, profesores disponibles y salones, número de materias que deben enseñar los profesores a clases específicas y un conjunto de restricciones. La terminología utilizada en la definición del problema difiere drásticamente de un estudio a otro [40].

- ***Programación de horarios de clase para Universidades*** o *University Course Timetabling Problem* (UCTTP): Es un problema de optimización híbrido en la clase de los problemas NP-duros, ocurre al inicio de cada semestre en las universidades e incluye la asignación de eventos (cursos y estudiantes) a un número fijo de periodos de tiempo y salones [10]. Básicamente hay dos formas de UCTTP: Horarios de clase basados en el currículo y Horarios de clase después de la inscripción.

Esta clasificación no es estricta, ya que hay problemas que pueden estar entre dos clasificaciones. El presente trabajo se centra en un caso particular de programación de horarios de clase para universidades basados en currículo, pero basado en previsiones, no en las inscripciones ya realizadas. En las secciones siguientes se contemplarán las definiciones más importantes para llevar a cabo el acometido.

Los *horarios de clase basados en el currículo* o *Currículum-based Course Timetabling Problem* (CB-CTT) por sus siglas en Inglés buscan construir horarios teniendo en cuenta los planes de estudio especificados por la universidad[42]. Toma como base el historial de inscripciones del semestre anterior y la cantidad de alumnos que aprobaron dichas. Por otra parte los *Horarios de clase después de la inscripción* o *Post-enrollment Course Timetabling Problem* (PE-CTT) se realizan una vez que los estudiantes han seleccionado las materias que van a cursar. Las restricciones del problema son especificadas por los datos de inscripción [42].

Dependiendo de la institución para la que se resolverá el problema de calendarización, es posible distinguir dos tipos de restricciones que determinan el grado de bondad de una solución.

2.1.1. Restricción fuerte o Hard constraint

Las restricciones fuertes son condiciones que deben ser satisfechas completamente en el problema para generar una posible solución sin ningún conflicto [10]. Se debe satisfacer esta restricción de manera obligatoria [42], por ejemplo:

- Varias clases asignadas a un mismo profesor en un solo tiempo
- Varias clases de diferentes profesores o grupos asignadas en el mismo espacio
- Clases asignadas fuera del horario de clases (clases de diferente duración).
- Las asignaciones se deben hacer en lugares con recursos suficientes para que se lleve a cabo la clase (pizarrón, proyector, computadoras, laboratorios, etc.).

2.1.2. Restricción débil o soft constraint

Las restricciones débiles son condiciones que están relacionadas con la función objetivo; la función objetivo es para maximizar el número de restricciones débiles satisfechas o, como en nuestro caso, minimizar el número de restricciones débiles insatisfechas. Estas restricciones no necesariamente requieren ser satisfechas, pero conforme se incrementa el número de satisfactibilidad, la calidad de la solución aumenta [10]. Es deseable que se satisfaga esta restricción, mas no es obligatorio [42], por ejemplo:

- Que una asignación sea en una hora determinada por la institución o por preferencia de los profesores.
- Los profesores necesitan tiempo para sus investigaciones por lo que pueden dar clases en horas continuas.

- un grupo de alumnos necesita ser asignado en un determinado lugar por razones de convivencia con otras carreras o por cercanía a la siguiente clase.

2.1.3. Complejidad

Todos los TTP comparten las mismas características básicas del problema general, sin embargo, la gran cantidad de variantes entre los diferentes problemas hacen que sea prácticamente imposible construir soluciones generales para todos los problemas. Debido a esto las investigaciones generalmente se especializan en un problema concreto o en un subconjunto de problemas con características similares.

En algunos casos, el problema de calendarización consiste en encontrar cualquier horario que satisfaga todas las restricciones. En estos casos, el problema es formulado como un *problema de búsqueda*. Por otro lado, el problema es formulado como *problema de optimización* cuando se requiere un horario que satisfaga todas las restricciones fuertes y minimice (o maximice) una función objetivo dado que incorpora las restricciones débiles.

En ambos casos, se define el *problema subyacente*, el cual es decidir si existe una solución, en el caso del problema de búsqueda; y el problema de decidir si existe una solución con valor dado de la función objetivo, en el caso de un problema de optimización [44]. El problema subyacente es **NP-COMPLETO** en casi todas las variantes. Sin embargo, existen soluciones exactas tan solo para casos pequeños (menos de 10 cursos), mientras que las instancias reales usualmente involucran cientos de cursos. La mayoría de los casos resueltos en tiempo polinomial no incluyen las restricciones mas comunes que aparecen en los problemas reales. Algunos métodos heurísticos son factibles, pero no garantizan que puedan encontrar la solución óptima.

Even, Itati y Shamir [22] demostraron que los TTP mas comunes son NP-completos aplicando una reducción del Problema de Coloración de Grafos (*Graph Colouring Problem*). En [15] mostraron que la NP-completitud aparece cuando cada estudiante puede elegir entre un conjunto de asignaturas y cuando las clases son de diferente duración.

Muchos autores creen que el problema de horarios no puede ser completamente automatizado, una de las razones es que existen horarios mejor

que otros que no pueden ser fácilmente expresados en un sistema automático. Por otra parte, el espacio de búsqueda es demasiado amplio, por esta razón es necesario la intervención de un humano que puede sesgar los resultados hacia el resultado deseado, mismo que el sistema no podría encontrar por sí solo[44]. Así mismo el sistema de la UTM debe ser un sistema de apoyo o sistema de recomendación, ya que no se garantiza un resultado completamente factible.

2.2. Heurísticas

En esta sección se aborda el tema de las heurísticas, que son el primer paso para adentrarnos en el mundo de las meta-heurísticas, además de que es importante saber porqué no se puede utilizar una heurística para resolver un problema de horarios de manera adecuada.
Las Heurísticas son procedimientos simples, a menudo basados en el sentido común, que se supone obtendrán una buena solución a problemas difíciles de un modo sencillo y rápido [54]. El término heurística proviene del vocablo griego heuriskein, que puede traducirse como encontrar, descubrir, hallar [39]. Según el mismo autor actualmente existen dos interpretaciones posibles para el término. La primera de ellas concibe las heurísticas como un procedimiento para resolver problemas. La segunda interpretación de heurística entiende que éstas son una función que permiten evaluar la bondad de un movimiento, estado, elemento o solución. Además sugiere dos clasificaciones para las heurísticas:

- Métodos constructivos: Construyen una solución al problema literalmente paso a paso desde cero. Generalmente, son métodos determinísticos y tienden a basarse en la mejor opción de cada iteración[16].
- Métodos de búsqueda: Comienza con alguna solución factible del problema e intenta mejorarlo progresivamente. Cada paso del procedimiento lleva a cabo un movimiento de una solución a otra con un mejor valor. El método termina cuando, para una solución, no hay otra solución accesible que lo mejore[16].

Las heurísticas pueden utilizarse para resolver el UCTTP, sin embargo suelen resolver el problema sólo parcialmente al ser más propensas a caer

en óptimos locales. Por esta razón y porque se requiere que se cumplan con las restricciones fuertes, no se pueden utilizar estos métodos para el problema de horarios de la UTM.

2.3. MetaHeurísticas

Tanto la adaptación al problema de horarios de la UTM, las implementaciones y los detalles de los algoritmos a utilizar para resolver el problema de horarios de la UTM se presentan en el siguiente capítulo, por lo cual en esta sección solo se presentan las definiciones y los algoritmos de manera general.

Las meta-heurísticas son otro tipo de algoritmos que permiten resolver el problema de UCTTP de manera mucho más óptima que las heurísticas o incluso utilizándolas para refinar la búsqueda hacia una solución más óptima, pueden estar categorizadas en búsqueda local y técnicas basadas en población[8]. Entre las meta-heurísticas de búsqueda local se encuentran: la Búsqueda Tabú[7], el Recocido Simulado[33], el Gran Diluvio[3], Búsqueda en Vecindario Variable (VNS)[2], entre otras. En las meta-heurísticas basadas en la población se encuentran: optimización por Colonia de Hormigas[46], Algoritmos Genéticos (GA)[53], Algoritmo de Búsqueda Armónica (HSA)[5], optimización por enjambre de partículas [30], etc.

2.3.1. Búsqueda Tabú (Tabu Search)

La Búsqueda Tabú (TS) es una Técnica de Búsqeda local diseñada para resolver problemas de optimización. Los orígenes de TS pueden situarse en diversos trabajos publicados a finales de la década de los 70. La Búsqueda Tabú es un procedimiento heurístico utilizado para resolver problemas de optimización de gran escala, el cual está diseñado para guiar a otros procedimientos de búsqueda local para escapar de óptimos locales y poder explorar mejor y más extensamente el espacio de soluciones. La búsqueda Tabú utiliza una memoria temporal para guardar la lista tabú y solo guarda las soluciones visitadas más recientemente. Estas soluciones son marcadas como “taboo” y previenen la re-evaluación de las mismas en el futuro [48]. En la Figura 2.1 se presenta el algoritmo.

```
procedure TABU_SEARCH
  s = s0 #Solucion inicial
  sBest = s
  tabuList = null
  while (not stoppingCondition()) #mientras no se cumpla
      condicion de parada
    candidateList = null
    Init sNeighborhood
    for(sCandidate in sNeighborhood)
      if(not containsTabuSolution(sCandidate, tabuList))
        candidateList = candidateList + sCandidate
      End
    end
    sCandidate = LocateBestCandidate(candidateList)
    s = sCandidate
    if(fitness(sCandidate) <= fitness(sBest))
      tabuList = tabuList + sCandidate
      sBest = sCandidate
      tabuList = tabuList-tabuList[maxTabuSize]
    end
  end
  return(sBest)
end_procedure
```

Figura 2.1: Algoritmo de Búsqueda Tabu

Para el algoritmo de Búsqueda Tabú se tienen en consideración los siguientes parámetros:

- *Iteraciones*: Número de veces que el algoritmo va a buscar una solución nueva a partir de la solución obtenida previamente.
- *Tamaño de memoria*: Tamaño de la lista tabú o memoria a corto plazo, que guarda las soluciones mejoradas dados los requerimientos del algoritmo.
- *Número de vecinos*: Determina el número de soluciones que se obtienen de la modificación de una solución base.
- *Porcentaje de modificación*: Número de elementos que se van a modificar de una solución para obtener un vecino.

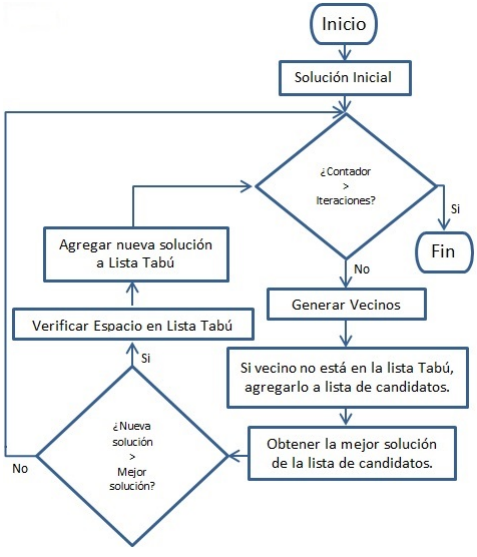


Figura 2.2: Diagrama de flujo del Algoritmo de Búsqueda Tabú.

Siguiendo el algoritmo de Búsqueda Tabú presentado en el diagrama de la Figura 2.2, el algoritmo se desarrolla de la siguiente manera:

1.- Se genera una **solución inicial** que va a ser el punto de partida en la búsqueda de una buena solución, de esta manera se le considera como mejor solución de inicio.

2.- Se crea una **Lista Tabú** vacía del tamaño de la memoria para que pueda ser utilizada durante el desarrollo del algoritmo, además se define la **condición de parada** (número de iteraciones) que tiene que cumplirse para llegar a encontrar la solución esperada.

2.1. **Se crean** tantos **vecinos** como lo indica el número de vecinos a partir de la solución actual, tomando en cuenta el porcentaje de modificación que se tiene como parámetro. Esto se hace para que las nuevas soluciones tengan información de la solución actual y no se modifiquen demasiado las nuevas soluciones. Además los valores para modificar la solución se generan a partir de los datos de entrada.

2.2. Se verifica que no estén los elementos de la vecindad en la lista Tabú, de esta manera ir agregando los elementos que no estén en esa lista a una lista de candidatos.

2.3. Se asigna una **solución candidata**, a partir de una búsqueda en la lista de vecinos de la solución actual. Esta solución candidata deberá tener el mínimo valor evaluado con la función de evaluación de todos los vecinos.

2.4. Si el valor de la función de evaluación de la solución candidata es menor al valor de la mejor solución evaluada con la función de evaluación, entonces a la lista Tabú se le agrega la solución candidata. También se toma como nueva mejor solución a la solución candidata. Por último se verifica que la lista Tabú aún tenga espacio; caso contrario, se eliminan elementos de la lista para mantener un espacio acorde al tamaño de la memoria.

2.5. Este procedimiento se repite desde el punto 2.1, y al final del ciclo se obtendrá la mejor solución encontrada para esa ejecución del algoritmo.

2.3.2. Búsqueda Armónica

El Algoritmo de Búsqueda Armónica (HS) es un algoritmo meta-heurístico relativamente nuevo desarrollado por Geem. Este algoritmo imita el proceso de improvisación musical en la cual un grupo de músicos reproducen los tonos de sus instrumentos musicales en conjunto buscando una armonía agradable según lo determinado por un estándar de audio-estética. Es considerado un algoritmo basado en población con aspectos de búsqueda local [6]. El algoritmo de Búsqueda Armónica se presenta en la Figura 2.3.

```
procedure ARMONIC_SEARCH
  HM = (n) Inicializar memoria armonica random con tamanio n
  while (not stoppingCondition()) #mientras no se cumpla
      condicion de parada
    #se puede elegir la nueva nota de dos maneras
        #1.-Random
        n = rand
        #2.-Con ajuste de tono
        if probabilidad:   #si se cumple con criterio de
            probabilidad
          n = HM[rand]     #se escoge al azar una nota de la
              memoria armonica
          n = modificar(n) #se ajusta el tono
        #Si la nueva armonia es mejor que la peor de la memoria
            armonica, sustituir peor por nueva
        p = peor(HM)
        if f(n) < f(p):
          p = n
  #Elegir la mejor armonia de la memoria.
  return best(HM)
end_procedure
```

Figura 2.3: Algoritmo de Búsqueda Armónica

Para el algoritmo de Búsqueda Armónica descrito en el diagrama de la Figura 2.4 se utilizan las siguientes variables:

- *Iteraciones*: Número de veces que el algoritmo va a buscar una solución nueva.
- *Tamaño de memoria armónica*: Tamaño de la memoria a corto plazo que guarda las soluciones.

- *Ancho de desplazamiento*: Número de cambios que se realizan en una solución, sustituyendo elementos de la representación de una asignación.
- *Razón de exploración*: Probabilidad para escoger entre la generación de una solución nueva o la obtención de la solución desde la memoria armónica.
- *Razón de ajuste de tono*: Probabilidad de modificación de una solución.

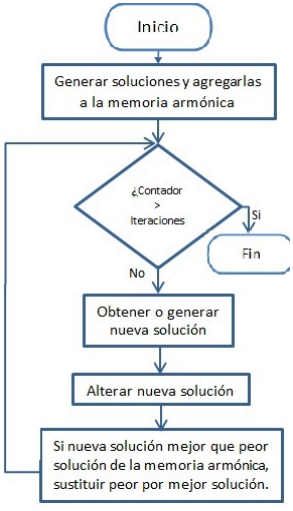


Figura 2.4: Diagrama de flujo del Algoritmo de Búsqueda Armónica.

Cada una de estas variables se utilizan en el proceso de desarrollo del algoritmo como se muestra a continuación:

1.- Al inicio se generan soluciones que se van a ir agregando a la **memoria armónica** hasta completar el tamaño de memoria armónica.

2.- En este punto se hace la verificación de la **condición de parada** (iteraciones), si se cumple hacer los siguientes puntos.

3.- Utilizando la razón de exploración, se verifica si un elemento nuevo se va a **obtener** de la memoria armónica de manera **aleatoria** o va a generar una **nueva solución**. En caso de que se obtenga de la memoria armónica, se va a modificar de acuerdo a la razón de **ajuste de tono** y al **ancho de desplazamiento** (parecido a la mutación en el algoritmo genético).

4.- Se escoge la **peor solución** de la memoria armónica y si el valor del elemento nuevo escogido en el punto anterior es mucho mejor, entonces **se sustituye** en la memoria armónica. Este proceso se repite desde el punto dos hasta completar el ciclo y al final quedarán las mejores soluciones en la memoria armónica.

2.3.3. Recocido Simulado

El Recocido Simulado (SA) es un método de búsqueda local inspirado en el calentamiento de sólidos en las ciencias de la física. Este enfoque evita quedar atrapado en el óptimo local y utiliza métodos de búsqueda local más fácilmente. Las soluciones obtenidas por la búsqueda local reemplazan la solución actual frecuentemente y este se repite hasta alcanzar alguno de los criterios de satisfacción. El proceso de inicialización del algoritmo es a través de la creación de una solución inicial aleatoria y, en cualquier iteración del algoritmo SA la solución actual es reemplazada con una solución aleatoria, la cual probablemente puede ser la solución óptima al problema. El proceso de búsqueda comienza con una temperatura alta; dicha temperatura se va decrementando de una manera progresiva y suave a lo largo de las iteraciones. Sin embargo, la velocidad del enfriamiento y la temperatura inicial son usualmente diferentes en cada problema, dependen de la experiencia que se tenga y la naturaleza del problema [40]. En la Figura 2.5 se puede apreciar el algoritmo.

Para el desarrollo del algoritmo de Recocido simulado se utilizan las siguientes variables:

- *Iteraciones*: Número de veces que el algoritmo va a buscar una solución nueva.
- *Temperatura inicial*: Valor inicial que tendrá la temperatura al iniciar el algoritmo, para generar soluciones variadas al inicio se puede

```
procedure SIMULATED-ANNEALING
  s = s0 # solucion inicial
  t = t0 # temperatura inicial
  while (not stoppingCondition()) #mientras no se cumpla
      criterio de parada
    begin
     generar probabilisticamente vecinos j de s
          if f(j) <= f(i): then
            aceptar j
          else:
            aceptar j con probabilidad exp((f(s)-f(j))/t)
    end
    decrementar t
  end
end_procedure
```

Figura 2.5: Algoritmo de Recocido Simulado

utilizar una temperatura alta.

- *Temperatura mínima*: Temperatura límite que se puede utilizar como condición de parada en caso de que la temperatura se decremente rápidamente o en caso de que se utilicen demasiadas iteraciones.

- *Decremento de temperatura*: valor que se resta de la temperatura en cada iteración, haciendo que la temperatura disminuya con cada iteración.

- *Número de vecinos*: Implica el número de soluciones que surgen de la modificación de una solución base.

- *Porcentaje de modificación*: Número de elementos que se modifican en una solución,sustituyendo elementos de la representación de una asignación.

Siguiendo el algoritmo descrito en el diagrama de la Figura 2.6, se utilizan los pasos que a continuación se mencionan:

1.- Primero se genera una **solución inicial**.

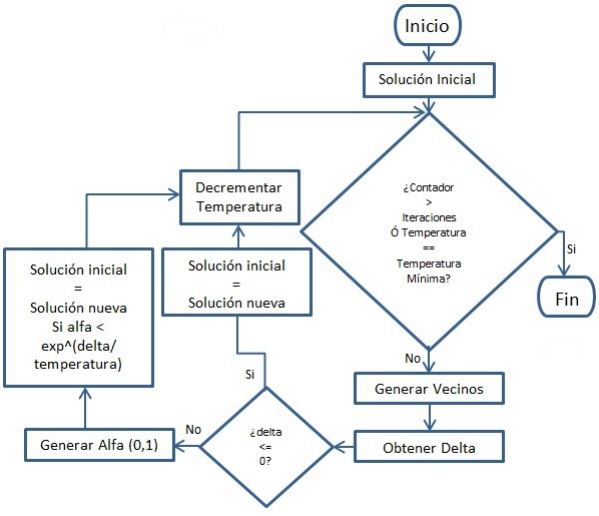


Figura 2.6: Diagrama de flujo del Algoritmo de Recocido Simulado.

2.- Hacer los siguientes puntos hasta que se cumpla con la **condición de parada** (iteraciones o estados):

2.1. Se obtiene una solución que se elige de **vecinos** creados a partir de la perturbación de la solución inicial, utilizando el **porcentaje de modificación** se generan vecinos similares.

2.2. Se calcula un **delta**, que es la diferencia entre el valor de la evaluación de la nueva solución obtenida en el paso anterior menos el valor de la evaluación de la solución inicial

2.3. Si el valor de delta es menor o igual a cero, entonces se asigna como solución inicial a la nueva solución generada en el paso 3, caso contrario, se genera una **valor aleatorio** en un rango de cero a uno que se compara luego con el valor de la **temperatura**; este valor se calcula utilizando el número de Euler o se puede utilizar alguna modificación ya que el desempeño del algoritmo radica mayormente en este valor. Si el valor generado de manera aleatoria es menor que el valor de la temperatura, entonces se

asigna como solución inicial a la nueva solución generada en el paso 2.1.

2.4. Por último **se decrementa la temperatura** utilizando el parámetro de decremento de temperatura y se repite el procedimiento desde el paso 2.

2.3.4. Algoritmos Genéticos (Genetic Algorithms)

Los Algoritmos Genéticos (GA) fueron introducidos primero por Holland[29] como un algoritmo robusto de búsqueda. Más tarde, especialmente en los trabajos de Goldberg fueron utilizados como técnica de optimización. Los Algoritmos Genéticos son métodos adaptativos que se basan en procesos de organismos biológicos, codificando una posible solución al problema en un cromosoma compuesto por una cadena de bits o de caracteres.

En los GA convencionales, los candidatos a soluciones frecuentemente se codifican en cadenas binarias, sin embargo, en el área de calendarización, las soluciones candidatas son usualmente codificadas en conjuntos porque éste agrega robustez a los operadores genéticos como la operación de cruzamiento [48].

Los algoritmos genéticos son un subconjunto de los algoritmos evolutivos y ésta se presenta a continuación en la Figura 2.7.

Para implementar el algoritmo genético, se necesita identificar cada uno de los componentes del algoritmo y asociarlos al problema de tal manera que se pueda utilizar en el proceso de búsqueda de solución. Podemos representar la solución entonces como objetos que son llamados individuos, y donde su codificación o cada una de las asignaciones que los componen son llamadas genotipos.

Partiendo de esta descripción y utilizando el algoritmo presentado en el diagrama de la Figura 2.8, el problema se desarrolla de la siguiente manera:

1.- Se **genera una población inicial**, que es un conjunto de soluciones (individuos) en un medio ambiente (número total de elementos en la población), y es la unidad que evoluciona en los algoritmos evolutivos,

```
procedure GENETIC_ALGORITHM
  1: Generar la poblacion Inicial.
  while (not stoppingCondition()) #mientras no se cumpla
      condicion de parada
    2: Evaluar la poblacion generada usando la funcion de
       evaluacion.
    3: Seleccionar algunas personas como los padres a cruzar
       basandose en la informacion obtenida de las funciones de
            evaluacion.
    4: Aplicar operador de cruzamiento para producir los hijos.
    5: Aplicar operador de mutacion a los hijos.
    6: Seleccionar a los padres e hijos para formar la nueva
       poblacion de la futura generacion.
end_procedure
```

Figura 2.7: Algoritmo Genético

de esta manera se puede obtener una diversidad en las soluciones de las generaciones siguientes.

2.- Una vez que se tiene la población inicial, **verificar** si se cumplen con las **condiciones de parada** (número de generaciones, solución fija, repeticiones constantes de una misma solución, etc.) para proceder a realizar los demás puntos descritos. En este caso se utiliza solo número de generaciones, ya que el algoritmo suele disminuir la función de evaluación en un número pequeño de generaciones y no suele disminuir en un valor cercano a cero.

2.1. Se **seleccionan las soluciones** de la población inicial (padres) con mejor valor de función de evaluación utilizando una selección por torneo, esto es, dependiendo de un porcentaje de selección se van filtrando las soluciones con menor valor de función de evaluación hasta generar el número requerido de individuos para la reproducción.

2.2. En este punto se realiza el **cruzamiento** de dos individuos a través del método N puntos-lineal; se escoge dos padres antes seleccionados y dependiendo de una probabilidad de cruzamiento, se realiza el cruce de esos dos elementos; y así sucesivamente

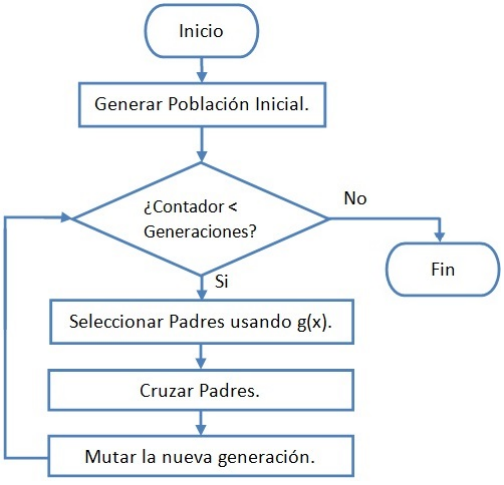


Figura 2.8: Diagrama de flujo del Algoritmo Genético.

con los demás padres hasta completar el la población que se considera para reproducirse en la siguiente generación.

2.3. Una vez que se cuenta con los individuos de la siguiente generación, se realiza una **mutación** de los elementos de la población utilizando una probabilidad de mutación para verificar si el elemento se va a mutar, en caso afirmativo se utiliza un porcentaje de mutación para modificar los valores de la solución de tal manera que sea mínimo el cambio, ya que la población actual es mejor que la población de la generación anterior.

2.4. Se **repite el procedimiento** desde el punto dos con la nueva población mutada y en la última generación se deben obtener soluciones con valor de función de evaluación mucho menor a la de las generaciones iniciales.

Siguiendo los puntos descritos, se utilizan las siguientes variables que se toman en consideración para implementar el algoritmo y que van a ser modificadas para variar los resultados del problema e intentar encontrar la mejor solución del mismo:

- *Número de generaciones*: El número de veces que el algoritmo va a generar nuevas y diferentes soluciones a partir de las ya existentes utilizando los métodos del algoritmo.
- *Tamaño de población*: Número de individuos que van a ser seleccionados, cruzados y mutados hasta completar una generación.
- *Porcentaje de selección*: Cantidad de individuos que son seleccionados como padres para generar la nueva generación.
- *Porcentaje de cruzamiento*: Cantidad de Cromosomas o Genes a intercambiar en un individuo (solución) del problema con otro individuo. Ambos deben haber sido seleccionados como padres previamente.
- *Probabilidad de mutación*: Posibilidad de modificar una pequeña cantidad de Cromosomas o Genes en un individuo (solución) aleatoriamente.
- *Porcentaje de mutación*: Cantidad de Cromosomas o Genes a modificar en un individuo durante el proceso de mutación.

2.4. Algoritmo de Descenso de Colina

En el algoritmo de Descenso de Colina 2.9, el algoritmo realiza una búsqueda de una nueva solución a partir de una solución inicial. Durante las iteraciones se va modificando en un porcentaje de la solución inicial, así se obtienen diferentes variedades de la misma y se guardan a una lista. Cuando ya no hay más iteraciones, se busca la solución con menor valor de función de evaluación dentro de la lista y se toma como el mejor en caso de que sea menor al valor de función de evaluación de la solución inicial.

2.5. Algoritmo de Descenso de Colina con Escalada Simple

El algoritmo de Descenso de Colina con escalada Simple 2.10 realiza una búsqueda de una nueva solución a partir de una solución inicial, durante las

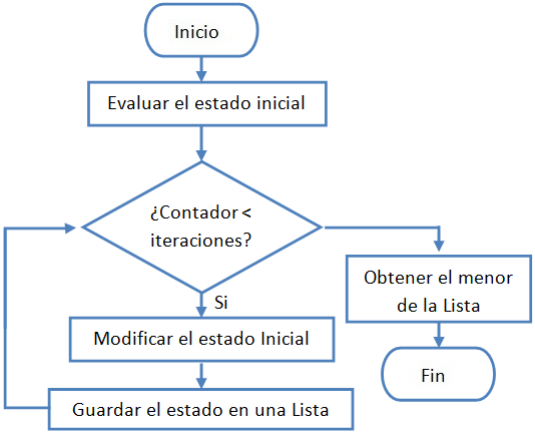


Figura 2.9: Algoritmo de Descenso de Colina.

iteraciones se va modificando en un porcentaje a la solución inicial, luego se evalúa ésta nueva solución en la misma iteración; si el valor de función de evaluación de la nueva solución es menor al de la inicial, se devuelve de inmediato este valor y termina el ciclo antes de que acaben las iteraciones.

2.6. Hiper-heurísticas

Además de las meta-heurísticas, es necesario definir que es una hiper-heurística y como funciona, de manera que se pueda apreciar este método como una alternativa para resolver el problema de horarios en la UTM. Cabe mencionar que no se ha encontrado bibliografía de resolución de problemas de horarios de una Universidad a través de una hiper-heurística utilizando un conjunto de meta-heurísticas sustituyendo al espacio de heurísticas.

El concepto de hiper-heurística se propuso por primera vez cuando fue necesario un algoritmo robusto que se pudiera generalizar y extender fácilmente a un problema nuevo pero aún similar [48]. Fue propuesto por Denzinger en 1996. Es un algoritmo de alto nivel que está compuesto por una

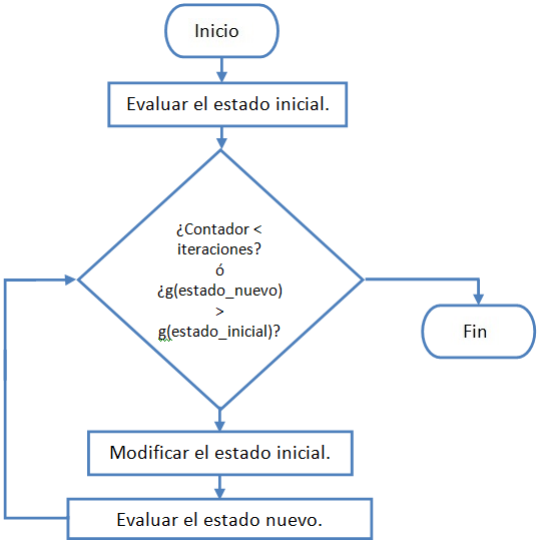


Figura 2.10: Algoritmo de Descenso de Colina con Escalada Simple.

cantidad "inmensa" de algoritmos heurísticos de bajo nivel. Las Hiper-heurísticas buscan el espacio de heurísticas en lugar del espacio de soluciones y, utilizan una limitada información de un problema en específico para controlar el proceso de búsqueda [13].

En [17] los algoritmos hiper-heurísticos se clasifican en:

1.- Hiper-heurísticas basados en la selección aleatoria de heurísticas de bajo nivel.

2.- Hiper-heurísticas codiciosos y hambrientos (*greedy* and *peckish*) que requieren una evaluación preliminar de todos o un subconjunto de las heurísticas a modo de seleccionar el que tenga mejor desempeño.

3.- Hiper-heurísticas basados en meta-heurísticas.

4.- Hiper-heurísticas empleando mecanismos de aprendizaje para manejar las heurísticas de bajo nivel.

La nueva clasificación según [13] son en dos categorías:

1.- *Selección heurística*: Estos enfoques construyen una solución de manera incremental. Empezando con una solución vacía, éstos inteligentemente seleccionan y utilizan heurísticas constructivas para construir gradualmente una solución completa.

2.- *Generación heurística*: La hiper-heurística busca un espacio de heurísticas construidas a partir de componentes en lugar de un espacio de heurísticas completas y predefinidas.

Una hiper-heurística puede escoger cuál heurística de bajo nivel aplicar en cada punto de decisión, hasta que un punto de parada sea satisfecha. Entonces se necesitan un conjunto de (meta)heurísticas y una función de evaluación para evaluar la calidad de la solución[**?**]. Se pueden utilizar diferentes tipos de selección, entre las que están la selección aleatoria para escoger el siguiente algoritmo a utilizar, la selección con mecanismo de aprendizaje, que guía a la hiper-heurística en la forma en que se seleccionan las (meta)heurísticas a utilizar o como se describe en [37], se puede asociar cada (meta)heurística bajo una condición del problema y por lo tanto, aplicar diferentes (meta)heurísticas en diferentes fases o partes del proceso de solución, tal como se muestra en la Figura 2.11.

```
Si (ProblemaTipo(P) = P1):
    Aplicar(metaHeuristica1,P)
Else if(ProblemaTipo(P) = P2):
    Aplicar(metaHeuristica2,P)
Else
```

Figura 2.11: Selección de la Hiper-Heurística

Algunos ejemplos de Hiper-heurísticas son: Algoritmos genéticos de alto nivel[25, 24, 27], programación genética lineal[52], Colonia de hormigas[18, 32], Búsqueda Tabú con aprendizaje[34, 31], entre otros. En el presente trabajo se implementa una hiper-heurística de tipo selección a partir las cuatro meta-heurísticas mencionadas anteriormente (Algoritmo Genético, Algoritmo de Búsqueda Tabú, Algoritmo de Búsqueda Armónica y el Algoritmo de Recocido Simulado) y se presenta detalladamente en el capítulo 3.

Capítulo 3

Problema de Horarios de la UTM

En este capítulo se aborda la representación del problema de horarios de la Universidad Tecnológica de la Mixteca, misma que es utilizada en el proceso de búsqueda de una solución por las meta-heurísticas y la hiper-heurística, con el fin de resolver el problema de manera adecuada. También se presentan las diferentes restricciones del problema, por último, el motivo por el cuál el problema se considera difícil de tratar. Así, a continuación se presentan los diferentes procesos que se realizan para obtener una representación adecuada de la solución.

3.1. Extracción de Datos

La Universidad Tecnológica de la Mixteca tiene un modelo Educativo en el que los alumnos deben tomar clases en un horario de 8:00 a 14:00 y de 16:00 a 19:00 horas, cinco días a la semana, teniendo horas de biblioteca y/o de sala de computación entre esas horas para que realicen las tareas o trabajos que se les asignan.
Además, la Universidad cuenta con una variedad de aulas que a la vez, cuentan con materiales didácticos indispensables para la impartición de clases hacia los alumnos, tales como pizarrón, plumones, proyector para presentaciones, computadoras, sillas o butacas, mesas, etc. Así, algunas

aulas son laboratorios que son designados para ciertas materias de una carrera en específico.
Los profesores normalmente laboran en un horario de 8:00 a 13:00 o 9:00 a 14:00 y de 16:00 a 19:00 horas. Es importante mencionar que varios profesores tienen diferentes comisiones o tareas en las que participan durante su jornada laboral, por lo cual no les es posible impartir clases a ciertas horas del día, estas comisiones o tareas generan más restricciones al problema de horarios.
Al momento de realizar el presente trabajo (ciclo escolar 2017-B), se consideran los siguientes datos de la Universidad Tecnológica de la Mixteca: 116 grupos, 237 profesores, 298 espacios y 1227 asignaturas.

En la siguiente Figura 3.1 se muestran los conjuntos de elementos que conforma el problema de Horarios de la UTM, donde M es el conjunto de Materias, P es el conjunto de Profesores, G es el conjunto de Grupos, E es el conjunto de Espacios, D es el conjunto de Días y H es el conjunto de Horas.

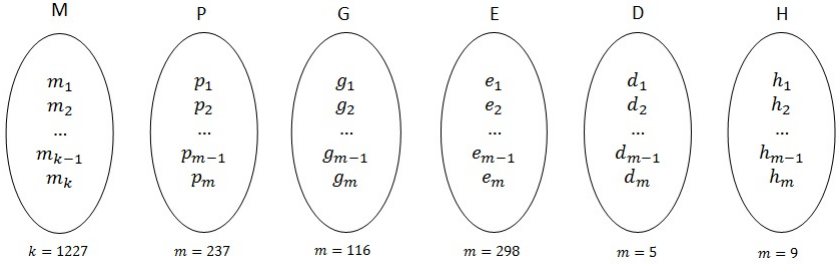


Figura 3.1: Conjuntos que conforman el problema de Horarios de la UTM.

Toda esta información se tiene capturado en una base de datos, la cual se modifica y/o se corrige en ocasiones.

La Figura 3.2 representa el proceso que se sigue para extraer, representar y utilizar los datos que se utilizan para resolver el problema de Horarios de la Universidad Tecnológica de la Mixteca.

A continuación, se describe brevemente el proceso de recolección de

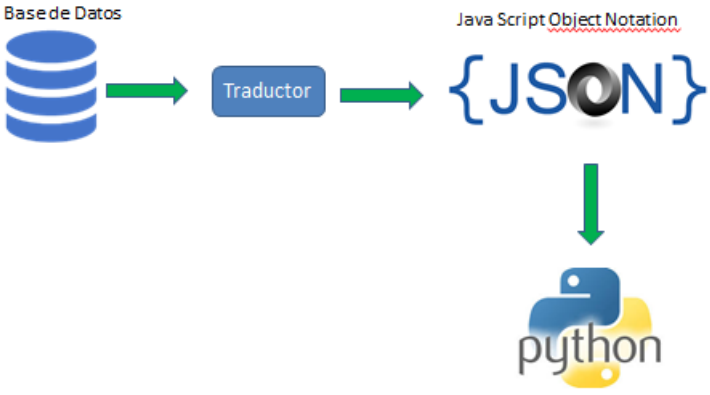


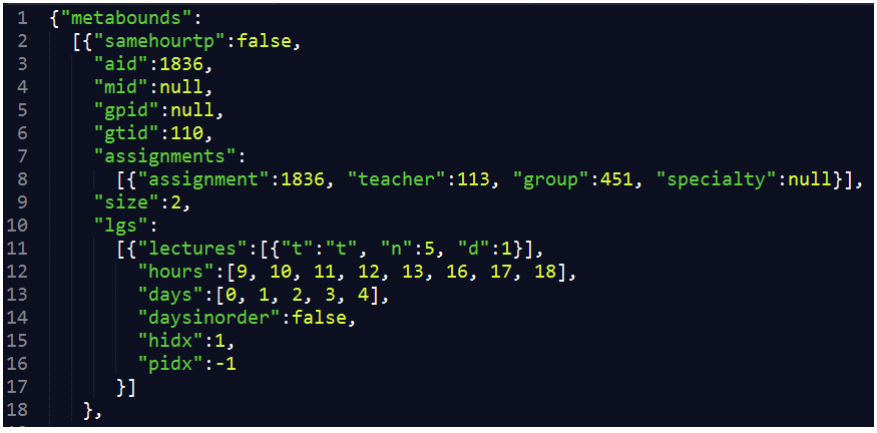
Figura 3.2: Obtención de datos para su uso en la implementación de las meta-heurísticas y la hiper-heurística.

datos, que es necesario para la generación de una representación de la solución, además este procedimiento se realiza en otro trabajo[1], por lo que no se profundiza en el tema.

La información, previamente capturado y guardado en una base de datos se parsea ó, en otras palabras, se traduce y se ordena de tal manera que pueda ser entendida y utilizada en un contexto simple. El parseo se realiza en un formato conocido como JSON (*Java Script Object Notation*), la cual facilita el intercambio de datos entre diferentes lenguajes de programación, tal es el caso con Python, además tiene la ventaja de ser legible y actualmente se ha convertido en un formato universal.
En este trabajo de Tesis, los datos traducidos en JSON se separan en dos bloques, una para manejar la información que se considera fija en este problema y el otro para manejar los datos dinámicos propios del problema; al primero se le llaman Metabounds (Figura 3.3) y al segundo se le llaman Bounds (Figura 3.4).

En el metabound se tiene vectores de *asignaciones* del tipo identifica-

[1]realizado por el Dr. José Figueroa



```
{"metabounds":
  [{"samehourtp":false,
    "aid":1836,
    "mid":null,
    "gpid":null,
    "gtid":110,
    "assignments":
      [{"assignment":1836, "teacher":113, "group":451, "specialty":null}],
    "size":2,
    "lgs":
      [{"lectures":[{"t":"t", "n":5, "d":1}],
        "hours":[9, 10, 11, 12, 13, 16, 17, 18],
        "days":[0, 1, 2, 3, 4],
        "daysinorder":false,
        "hidx":1,
        "pidx":-1
      }]
  },
```

Figura 3.3: Ejemplo de un Metabound del problema.

dor:valor, donde el identificador es un string y el valor puede ser cualquier tipo de dato que pueda ser procesado en JSON. Primero, se tiene un identificador principal que identifica a los elementos como metabounds, luego, el valor que contiene es el arreglo de asignaciones que a su vez contiene diferentes campos de las asignaciones, tales como materias profesores y grupos; estos campos están ligados de acuerdo a los criterios con las que se capturaron en la base de datos, por ejemplo, un profesor X tiene asignado una materia Y, además la materia Y está asignado a un grupo Z. Estas pertenencias se toman como datos fijos debido a la naturaleza del problema de Horarios. También en el ejemplo de la Figura 3.3 se puede observar un arreglo de *lgs*, de este dato se tienen las *lectures*, que contiene información tal como el número de días que se va a tener la asignación y la duración en horas de cada asignación.

A continuación se describen de manera mas detallada las variables principales utilizadas en el metabound de la Figura 3.3 y como influyen éstas para generar una asignación en el vector solución:

- En la línea 1 se muestra el identificador que denota como **metabounds** a los elementos que están dentro del arreglo que contiene

dicho identificador.

- En la línea 2 se indica si las clases de teoría y práctica inician a la **misma hora**, puede ser — "true.º "false".
- En la línea 7 se especifican los conjuntos de asignaciones que pueden verse por separado en la línea 8, las cuales son, el grupo que va a llevar una materia asignada a un profesor dado. También se incluye la especialidad en caso de que sea necesario que el mismo profesor le imparta clases a otro grupo de la misma especialidad. Cada valor de éstos elementos son enteros que están ligados a una representación física del identificador.
- En la línea 9 se define el tamaño del vector de la asignación actual. Es importante definir el tamaño para facilitar la obtención de las posibles asignaciones. El valor del identificador **size** es de tipo entero.
- En la línea 10 se enuncian las preferencias y necesidades de los profesores, las cuales son, el número de días que se va a impartir la materia, la duración de la materia a impartir y un identificador de si es una materia de teoría o de práctica. El valor del identificador **t** puede ser un string "t" o "p", la primera representando la clase teórica y la segunda la clase práctica.
- En las líneas 12 y 13 se listan las horas y los días en que el profesor puede impartir sus asignaciones.

Los bounds son una estructura de datos que se obtienen a partir del problema de horarios de la UTM y representan los datos de entrada que se utilizan para formar cada asignación de la solución.

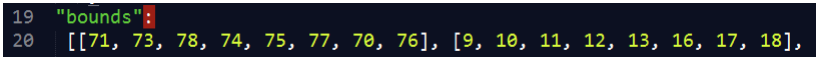


Figura 3.4: Ejemplo de un Bound del problema.

En los bounds se tienen diferentes vectores con los datos de Espacio, Hora y Día, estos datos están listados según se requiera en cada asignación,

por ejemplo, se puede requerir que se tenga días diferentes para clases de práctica y teoría, entonces se debe tener arreglos de Espacio, Hora Teoría, Día Teoría, Hora Práctica, Día Práctica. Estos requerimientos vienen incluidos en los metabounds. Se puede apreciar que en el ejemplo, en la línea 19 se define el identificador principal como "Bounds", luego en la línea 20 empieza su estructura, en este caso se muestran dos vectores (vector de espacios y vector de horas pertenecientes al metabound de la Fig. 3.3). Estos son los únicos datos que se pueden escoger para formar la primera asignación de la solución, restringidos por el metabound correspondiente.

Es necesario aclarar que los bounds y metabounds se generan una sola vez o cada vez que haya cambios en la base de datos que sean considerados necesarios en la resolución del problema, es así como se ahorra un proceso que puede afectar el tiempo de ejecución en los algoritmos, si los datos se obtuvieran directamente de la base de datos.

3.2. Diseño de la solución

Una solución base al problema (Figura 3.5), es el conjunto de asignaciones que se deben realizar para construir el horario de clases, estas asignaciones deben satisfacer ciertas restricciones del problema. La solución base no necesariamente tiene satisfechas todas las colisiones que se generan, por lo que aún necesita procesarse por algún algoritmo para mejorar su contenido. En el ejemplo presentado se tienen n asignaciones, que es el número de materias que se cursan en la Universidad, también se puede visualizar que cada asignación se conforma por elementos de Materia, Profesor, Grupo, Espacio, Día y Hora.

A continuación se describen los elementos que conforman una asignación y que a su vez se complementan para generar una solución base.

- Los *Espacios* representan salones de clases o aulas especiales (cubículos, salas de cómputo) en la UTM que cuentan con los elementos necesarios para que sea posible impartir una clase dentro de ella. Los espacios se denotan por E, tal que $E = \{1, ..., 95\}$.

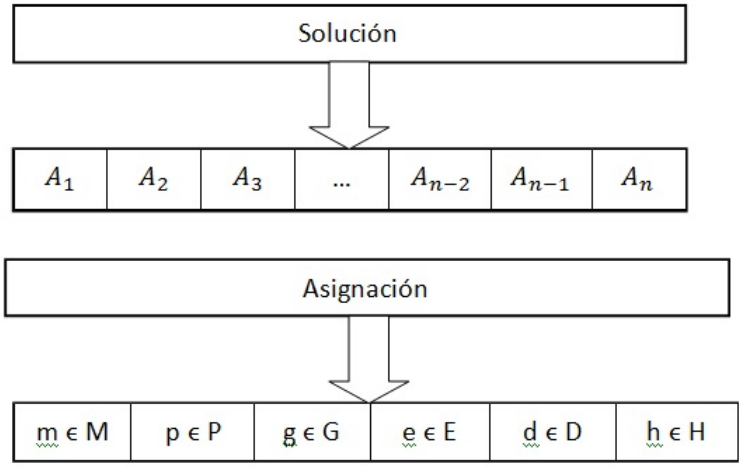


Figura 3.5: Ejemplo de una solución base.

- Los *Días* son los días hábiles de la semana en las que es posible impartir clases, están denotados por D, tal que $D = \{1, ..., 5\}$.

- Las *Horas* son el tiempo del día en las que es posible asignar el inicio de una clase; están denotadas por H, tal que $H = \{1, ..., 9\}$.

- Los *Profesores* son los catedráticos de tiempo completo que laboran en la Universidad y a los que se les puede asignar alguna materia a impartir; se denotan por P, tal que $P = \{1, ..., 200\}$.

- Las *Materias* se refieren a cada una de las asignaturas de los planes de estudios que se dictan en la Universidad Tecnológica de la Mixteca (Cálculo Diferencial, Programación orientada a objetos, Comunicación Visual, etc.); y se denotan como M, tal que $M = \{1, ..., 246\}$.

- Por último los *Grupos* están formados por un conjunto de alumnos que cursan una misma carrera en un semestre determinado, se deno-

tan por G, tal que $G = \{1, ..., 50\}$.

Se dice que existe una asignación, si el vector que la representa está conformado por un espacio $e \in E$, una duración $d \in D$, una hora $h \in H$, un profesor $p \in P$, una materia $m \in M$ y un grupo $g \in G$, esto es:

$$a(< e, d, h, p, m, g >) \tag{3.1}$$

Luego, una solución está compuesta por diferentes asignaciones:

$$S = [a_1, a_2, ..., a_n] \tag{3.2}$$

tal que n es el número de asignaciones que requiere la UTM durante dicho semestre.
Además definimos las características principales que forman parte de una asignación y que son necesarias para establecer las restricciones de manera adecuada.

- La duración, es el tiempo que se impartirá una clase, 1 hora como mínimo y 5 horas como máximo; se denota por du, donde $du = \{1, ..., 5\}$.
- La hora de inicio es el tiempo en que debe iniciar una clase, está acotada desde las 8:00 hasta las 13:00 horas y luego desde las 16:00 horas hasta las 18:00 horas, se denota por hi, donde $hi = \{8, ..., 13\} \cup \{16, ..., 18\}$.
- La hora de Fin se refiere a la hora en que debe terminar una clase, está entre los intervalos 9:00 hasta las 14:00 horas y de 17:00 horas hasta las 19:00 horas; se denota por hf, donde $hf = \{9, ..., 14\} \cup \{15, ..., 17\}$.
- La clase de Teoría se refiere a si existe una clase teórica en la asignación, se denota por ta, donde $ta = \{1, 0\}$.
- La clase de Práctica se refiere a si existe una clase práctica en la asignación, se denota por pa, donde: $pa = \{1, 0\}$.

En este trabajo de Tesis, se utiliza una representación de la solución, en vez de la solución completa como la que se muestra en la Ecuación 3.2. En

la Figura 3.6 se muestra la estructura de la representación de la solución, en donde solamente se tienen contemplados tres tipos de datos: Espacio, Día y Hora; éstos datos se ligan con los datos faltantes a través del Metabound para conformar una solución completa.

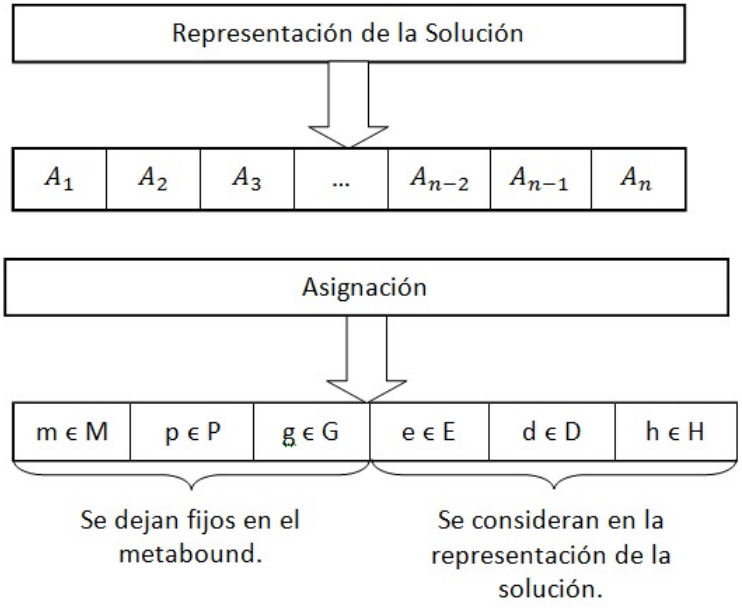


Figura 3.6: Ejemplo de la representación de la solución.

La representación de la solución está formada por n asignaciones, donde n es el número de asignaciones que requiere la UTM durante un semestre. Luego cada asignación de la representación de la solución está conformada por 3 a 6 elementos de diferentes tipos: para los espacios y horas, se contemplan valores enteros y para los días, se considera un vector de enteros en un rango de 0 a 4, donde 0 representa Lunes y 4 Viernes.

La representación de la Solución está ligado fuertemente con los bounds y los metabounds, las cuáles filtran los datos elegibles para cada asignación. Por ejemplo, en la Figura 3.7 se puede observar un ejemplo de una solución, en donde se muestra encerrado en color morado la primera asignación, ésta

asignación contiene dos números enteros, el primero representa un valor de Espacio y el segundo un valor de Hora, éstos datos se eligen aleatoriamente de los bounds, además se puede ver que el primer metabound tiene especificado el "size":2, lo cual significa que tal asignación en específico debe contener dos elementos obligatoriamente, independientemente de las demás asignaciones anteriores y siguientes.

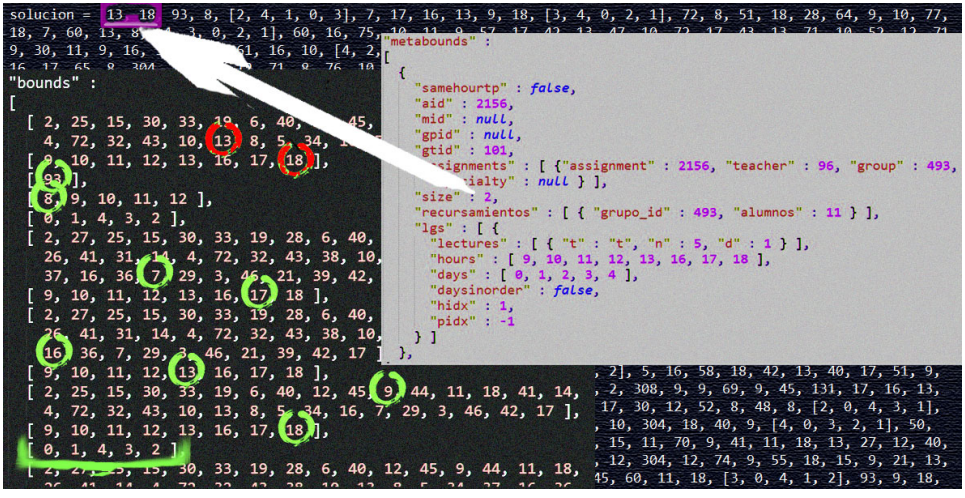


Figura 3.7: Ejemplo de la primera asignación en la Representación de la Solución.

Con esta representación se busca agilizar el proceso que realizan los algoritmos Meta-heurísticos e Hiper-heurístico que se presentan en el trabajo de Tesis y durante el proceso de ejecución de estos algoritmos, existe una parte en la que se genera o se modifica una solución, este procedimiento implica aleatorizar los datos que van a ser agendados dinámicamente. Así, durante la ejecución del algoritmo se puede ahorrar una porción del tiempo del que se tendría si se consideran todos los datos del problema en el proceso de generación y modificación de la solución. Como punto importante, se menciona que en la evaluación se deben considerar todos los datos del problema (Espacios, Profesores, Asignaturas, Días, Horas y Grupos) para verificar las colisiones en la solución. Es por eso que por motivos de co-

modidad, a la representación de la solución (vector solución con datos de Espacios, Horas y/o días) se le puede decir solución, sin olvidar que depende de otros datos para estar completo.

Por último, el vector solución (representación de la solución) procesado con los algoritmos que se presentan en este trabajo, es utilizado para generar los horarios de manera visual en el sistema.

3.3. Restricciones

Las restricciones, como se describe en el capítulo de marco teórico, se clasifican en fuertes y débiles. Las restricciones fuertes que se consideran para la UTM son 9 (13 en total). Cabe mencionar que la restricción número 11 referente al horario del alumno en recursamiento no se tomará en cuenta por el momento debido a que no se cuenta con dicha información en el sistema. Las restricciones fuertes totales son las siguientes (en cursivas las que no se consideran o las que se pueden mezclar para reducir la complejidad):

1.- *Las clases de teoría no deben ser el mismo día, a la misma hora.*

2.- *Las clases de práctica no deben sean el mismo día, a la misma hora.*

3.- Las clases de teoría y práctica no deben ser a la misma hora el mismo día.

4.- *Las clases no deben impartirse durante algún día restringido para alguna de ellas.*

5.- *Las clases no deben impartirse durante una hora restringida para ellas.*

6.- Las clases no deben impartirse durante algún día y hora restringido para alguna de ellas.

7.- Un profesor no puede estar en dos lugares (clase) distintos a la misma hora, el mismo día.

8.- Dos o más profesores o grupos no pueden estar en el mismo lugar (clase) a la misma hora, el mismo día. A menos que sea una multi-asignación de grupos y/o profesores.

9.- Un mismo grupo no puede tener asignada otra clase, ya sea en el mismo o diferente lugar a la misma hora, el mismo día. A menos que sean clases de diferentes especialidades de ese grupo.

10.- Si un profesor da clases a las 8:00 a.m., no puede dar clases a la 1:00 p.m., a menos que se indique explícitamente lo contrario en los metabounds.

11.- Un grupo no puede tomar clases obligatorias a la hora en la que un alumno que pertenece al mismo grupo deba tomar clases de recursamiento junto a otro grupo.

12.- Un grupo debe tener dos horas libres al día para que en alguna de ellas se les pueda asignar clases de inglés. A menos que se indique lo contrario a través de la definición de metabounds de sus respectivas asignaturas. Esto ocurre en al menos tres semestres de la Ingeniería en Alimentos y dos de la Ingeniería en Diseño.

13.- Un espacio solo puede ser usado oficialmente 9 horas al día, ya sea por el mismo o diferentes grupos.

Algunas restricciones fuertes se consideran de manera implícita dentro de la estructura de los MetaBounds en JSON, tal es el caso de la restricciones uno y dos, por este motivo, se considera como una sola restricción (número 3). Para las restricciones 4 y 5, se consideran en los bounds como los datos que pueden ser elegidos, por lo que se toma como una única restricción (número 6) y solamente se evalúa que no se salga de horario alguna de los valores ingresados en los bounds. La restricción número 11 no se considera por el momento por falta de información necesaria para su evaluación.
Es así como se tienen las siguientes restricciones fuertes considerados en el problema:

1.- Las clases de teoría y práctica no deben ser a la misma hora el mismo día.

2.- Las clases no deben impartirse durante algún día y hora restringido para alguna de ellas.

3.- Un profesor no puede estar en dos lugares (clase) distintos a la misma hora, el mismo día.

4.- Dos o más profesores o grupos no pueden estar en el mismo lugar (clase) a la misma hora, el mismo día. A menos que sea una multi-asignación de grupos y/o profesores.

5.- Un mismo grupo no puede tener asignada otra clase, ya sea en el mismo o diferente lugar a la misma hora, el mismo día. A menos que sean clases de diferentes especialidades de ese grupo.

6.- Si un profesor da clases a las 8:00 a.m., no puede dar clases a la 1:00 p.m., a menos que se indique explícitamente lo contrario en los metabounds.

7.- Un grupo debe tener dos horas libres al día para que en alguna de ellas se les pueda asignar clases de inglés. A menos que se indique lo contrario a través de la definición de metabounds de sus respectivas asignaturas. Esto ocurre en al menos tres semestres de la Ingeniería en Alimentos y dos de la Ingeniería en Diseño.

8.- Un espacio solo puede ser usado oficialmente 9 horas al día, ya sea por el mismo o diferentes grupos.

A continuación se muestran las restricciones débiles que se tienen en la UTM:

1.- Que los profesores tengan un horario continuo de clases.

2.- Que los alumnos tengan un horario continuo de clases. Pero siempre con horas asignadas en la tarde y en la mañana.

3.- Que no existan clases de práctica fuera del horario habitual de clases, a menos que se especifique directamente en los datos de entrada.

4.- Que los profesores impartan las mismas materias que en semestres anteriores afines.

5.- Que los profesores tengan horarios muy similares, si no es que el mismo que en semestres anteriores afines.

6.- Que los grupos no tomen clases continuas en lugares demasiado alejados.

7.- Que los grupos de diferentes carreras tomen clases en grupos de aulas ïnterdisciplinarios".

Por falta de datos en el momento de la implementación, las restricciones débiles de la UTM que se abordan en la tesis para definir la función heurística de este problema son las siguientes:

1.- Que los profesores tengan un horario continuo de clases, a menos que el profesor especifique lo contrario. Y que tengan su horario en las horas que ellos prefieran.

2.- Que los alumnos tengan un horario continuo de clases. Pero siempre con horas asignadas en la tarde y en la mañana.

3.4. Función de Evaluación

La función de evaluación sirve para verificar la calidad de una solución, en el caso del problema de horarios de la UTM, para verificar que el vector solución cumpla con las restricciones dadas, obteniendo así un resultado libre de colisiones en caso de que la evaluación sea satisfactoria.
Se tienen entonces dos tipos de evaluación, una para verificar que se genere una solución factible (que no se solapen las asignaciones de una solución, que no se generen asignaciones fuera de la jornada de trabajo, que no existan dos asignaciones al mismo tiempo, entre otros) y otro para verificar que la solución cumpla con las necesidades del profesor y de la institución educativa.
Los dos tipos de evaluación pertenecen a los dos tipos de restricciones, primero está la función de factibilidad que se utiliza para evaluar las restricciones fuertes y segundo, la función heurística, que se utiliza para evaluar las restricciones débiles. Así, para obtener la evaluación final o total de una solución se agregará el valor de ambas funciones.

Se pretende obtener una función de evaluación que contabilice de manera adecuada las restricciones insatisfechas. Cada restricción, al ser evaluado genera un valor que se le llamará **costo de la restricción**.

3.4.1. Función de Factibilidad

Dado que la función de factibilidad pertenece a las restricciones fuertes, ésta se debe satisfacer de manera obligatoria o en el peor de los casos, obtener un valor cercano a cero. La función de factibilidad se define como sigue:

$$f(x) = \sum_{i=1}^{9} (peso_{x,i} * costo_{x,i}) \tag{3.3}$$

donde:
x es una asignación en la solución,
$costo_i$ corresponde al costo según el grado de insatisfacción de la restricción i,
$peso_i$ es el ponderador asociado a la i-ésima restricción fuerte que depende de la importancia de satisfacer la i-ésima restricción.
Para encontrar una solución que cumpla con los requerimientos de la Universidad, se necesita una solución que al evaluarla con la función de factibilidad dé un resultado igual a cero. Para esto se debe minimizar esta función en el proceso de búsqueda de la solución mediante algún método. Sin embargo, no siempre es posible minimizar por completo la función, ya que muchas veces existen incoherencias en los datos de entrada o no hay información suficiente para generar todas las restricciones; entonces se requiere obtener:

$$min(f) \approx 0 \tag{3.4}$$

En la Sección de Diseño de la Solución se tienen las definiciones que se utilizan para definir las restricciones. El grado de insatisfacción de cada restricción se define de la siguiente manera:

1.- Las clases de teoría y práctica no deben ser a la misma hora, el mismo día.

$$costo_1 = \sum_{i=1}^{m} TP_i \tag{3.5}$$

donde:
m = número de clases de Teoría y Práctica que son a la misma hora, el mismo día.

2.- Las clases no deben impartirse fuera del horario de clases.

$$costo_2 = \sum_{i=1}^{m} F_i \tag{3.6}$$

donde:
m = número de clases fuera del horario laboral de la UTM.

3.- Un profesor no puede estar en dos lugares (clase) distintos a la misma hora, el mismo día.

$$costo_3 = \sum_{i=1}^{m} PE_i \tag{3.7}$$

donde:
m = número de profesores que sí están asignados a dos lugares distintos, a la misma hora, el mismo día.

4.- Dos o más profesores o grupos no pueden estar en el mismo lugar (clase) a la misma hora, el mismo día. A menos que sea una multiasignación de grupos y/o profesores.

$$costo_{4_1} = \sum_{i=1}^{m} EP_i \tag{3.8}$$

donde:
m = número de profesores asignados a un mismo lugar y que no tienen alguna multiasignación.

$$costo_{4_2} = \sum_{j=1}^{n} EG_j \tag{3.9}$$

donde:
n = número de grupos asignados a un mismo lugar y no tienen alguna multiasignación.
$costo_4 = costo_{4_1} + costo_{4_2}$

5.- Un mismo grupo no puede tener asignada otra clase, ya sea en el mismo o diferente lugar a la misma hora, el mismo día. A menos que sean clases de diferentes especialidades de ese grupo.

$$costo_5 = \sum_{i=1}^{m} GE_i \tag{3.10}$$

donde:
m = número de grupos asignados a dos o más lugares y que no tienen especialidad o son de la misma especialidad.

6.- Si un profesor da clases a las 8:00 a.m., no puede dar clases a la 1:00 p.m.

$$costo_6 = \sum_{i=1}^{m} P_i \tag{3.11}$$

donde:
m = número de profesores que tienen clases a las 8:00 a.m. y también tienen clases a la 1:00 p.m.

7.- Un grupo debe tener dos horas libres al día para que en alguna de ellas se les pueda asignar clases de inglés.

$$costo_8 = \sum_{i=1}^{m} G_i \tag{3.12}$$

donde:
m = número de grupos que tienen menos de dos horas libres al día.

8.- Un espacio solo puede permitir 9 horas al día, ya sea del mismo o diferentes grupos.

$$costo_9 = \sum_{i=1}^{m} E_i \tag{3.13}$$

donde:
m = número de espacios que tienen más de 9 horas de asignaciones en un mismo día.

3.4.2. Función Heurística

La función heurística está compuesta por la suma del grado de insatisfacción de cada una de las restricciones débiles, ponderadas según su importancia.

$$h(x) = \sum_{i=1}^{2} (peso_{x,i} * costo_{x,i}) \tag{3.14}$$

donde:
$costo_i$ corresponde al costo según el grado de insatisfacción de la restricción i,
$peso_i$ es el ponderador asociado a la i-ésima restricción débil.

En esta función, el total del peso depende de las variables asociadas a la i-ésima restricción débil. En este caso, se busca minimizar el valor de la solución con respecto a la función objetivo, por lo que la modelación queda de la siguiente manera:

$$min \sum_{i=1}^{2} (peso_i * costo_i) \tag{3.15}$$

A continuación se numeran los costos de la función heurística, de tal manera que puedan ser utilizados en el algoritmo meta-heurístico. Es importante mencionar que por el momento sólo se utilizan dos de las restricciones débiles ya que aún no se cuenta con los datos necesarios para definir las demás restricciones. Los costos de la función heurística son:

1.- Que los profesores tengan un horario continuo de clases.

$$costo_1 = \sum_{i=1}^{m} P_i \tag{3.16}$$

donde
m = número horas en la que un profesor tiene asignaciones no continuas (se permiten asignaciones de clases que estén separados por más de 3 horas).

2.- Que los alumnos tengan un horario continuo de clases. Pero siempre con horas en la tarde y en la mañana.

$$costo_2 = \sum_{i=1}^{m} G_i + \sum_{j=1}^{n} G_j \tag{3.17}$$

donde
m = número de veces en que un grupo tiene asignaciones no continuas (se permiten asignaciones de clases que estén separados por más de 3 horas).
n = número de veces que un grupo no tiene asignaciones tanto en la mañana como en la tarde.

Tanto para la función de factibilidad como para la función heurística, se consideran de momento un peso de uno debido a que el trabajo pretende comparar los diferentes algoritmos. De manera que en un futuro sea posible modificar cualquier valor que se crea necesario y mejorar así los resultados. Utilizando las funciones de Factibilidad y la función Heurística, se define la función de evaluación de la siguiente manera:

$$g(x) = f(x) + h(x) \tag{3.18}$$

en donde: x es una solución.

3.5. Complejidad

Como se menciona en el Capítulo 2, el problema base de la calendarización de Horarios es NP-COMPLETO[19], lo cual significa que es un proble-

ma complejo en el ámbito computacional, tanto así que para éste problema específicamente un algoritmo exacto no es capaz de encontrar un buen resultado en un tiempo corto debido a que existen muchas soluciones[23]. En la tesis presentado en [36], se demuestra matemáticamente que el problema en sí mismo es NP-COMPLETO y al aplicarse a problemas reales, se convierte en un problema NP-DURO.

La cantidad de soluciones que se tiene en el problema de Horarios de la UTM se puede calcular a partir de los datos generados en los bounds, ya que éstos datos representan datos que deben ser utilizados para hacer una asignación y el conjunto de asignaciones forman una solución. Entonces tomando como punto de partida los bounds, se realiza el cálculo del número de soluciones que se pueden generar a partir de éstos datos. Si se realiza un conteo de los vectores en los bounds, se pueden obtener el número de elementos disponibles para cada asignación, por ejemplo:

$bounds = [[e_1 1, e_1 2, e_1 3, e_1 4, e_1 5], [h_2 1, h_2 2, h_2 3], [e_3 1, e_3 2], [h_4 1, h_4 2, h_4 3, h_4 4, h_4 5, h_4 6], [d_5 1, d_5 2, d_5 3, d_5 4, d_5 5]]$

A partir de estos datos se tiene el siguiente conteo de posibilidades
$bounds = [5, 3, 2, 6, 5]$
Éstos valores representan el número de posibilidades que son elegibles para generar una solución, por lo tanto para obtener el total de soluciones se multiplican entre sí, siguiendo el ejemplo:

$$numeroSoluciones = 5 * 3 * 2 * 6 * 5$$

$$numeroSoluciones = 900$$

En el problema de horarios de la UTM, el número de soluciones es de $7{,}8 \times 10^{1057}$, esta cifra representa todas las soluciones posibles, tanto factibles como no factibles. Sin embargo, se vuelve complejo encontrar la solución por el hecho de que en el problema de horarios de la UTM se tienen varias restricciones fuertes, mismas que se deben cumplir para que se pueda obtener un resultado factible y en ocasiones, los valores de las variables que representan dichas restricciones invalidan la misma. Además, no se puede

calcular una solución de manera directa en un tiempo relativamente corto, debido a que se tiene una cantidad enorme de soluciones y por el tamaño de los datos involucrados en el problema.

Capítulo 4

Comparativa de Meta-Heurísticas

Hasta este punto, se han definido tanto la función de factibilidad como la función heurística. Ésto permite que se puedan implementar los algoritmos meta-heurísticos, ya que éstos utilizan la función de evaluación en su proceso de búsqueda de una solución factible. Es así, como en el presente capítulo, se describe la implementación de los cuatro algoritmos seleccionados; seguida de las experimentaciones pertinentes que comprueban un buen desempeño de los mismos, en cuanto a función de evaluación y tiempo, para solucionar el problema de horarios de la UTM en el periodo 2017-B.

Cabe recordar que el objetivo de la meta-heurística, si bien es encontrar una muy buena solución dentro del espacio de soluciones posibles, no es encontrar el óptimo global.
Y, en el caso particular de esta aplicación, el objetivo particular es proponer diferentes horarios que sirvan como punto de partida a los encargados humanos de los mismos, para posteriormente modificar el elegido y adaptarlo a los requerimientos cambiantes que puedan ser más particulares de cada situación.

4.1. Introducción

En esta sección se detalla el procedimiento que se lleva a cabo para realizar la implementación de los algoritmos meta-heurísticos, incluyendo las diferentes variables necesarias para los algoritmos, la manera de cómo se utiliza la función de evaluación para calcular el coste de una solución y las modificaciones realizadas, en algunos casos, al algoritmo original.

A parte de describir la implementación de los cuatro algoritmos, se puntualizan las experimentaciones realizadas con los algoritmos propuestos, con el fin de obtener los diferentes resultados que hagan posible la comparación y mejora del algoritmo en la fase de diseño hiper-heurístico.

Esta experimentación se realiza en el lenguaje de programación Python y por cada algoritmo meta-heurístico se realizan 50 ejecuciones modificando los parámetros de entrada, de tal forma que sea posible obtener diferentes valores incluyendo la mejor combinación de parámetros para cada algoritmo.

Éstos parámetros serán los utilizados posteriormente en el algoritmo hiper-heurístico ya que de cada algoritmo se pretenden obtener los valores que generen un menor valor de función de evaluación con respecto al problema de horarios.

Con el fin de mostrar el desempeño de cada algoritmo para resolver el problema de horarios de la UTM, se comparan los resultados obtenidos utilizando diferentes combinaciones de parámetros por cada algoritmo, el procedimiento es la siguiente:

- Se realizan 50 ejecuciones por cada combinación de parámetros de entrada.
- Se filtran los resultados por tiempo de ejecución mínima, tiempo de ejecución promedio, valor de función de evaluación mínima y valor de función de evaluación promedio mínima.
- Se llega a concluir que el algoritmo del que se obtenga un valor de función de evaluación promedio menor, es el que, en teoría, tendría un mejor desempeño en cuanto a los otros algoritmos.

El tiempo de ejecución promedio muestra la cantidad de tiempo que es necesario invertir en un algoritmo para obtener un resultado, depende de las combinaciones de parámetros y es de ayuda al momento de decidir qué algoritmo utilizar en el sistema.

Cabe mencionar que para las experimentaciones realizadas, se utilizó una computadora de escritorio con las siguientes especificaciones:

- Procesador: Intel Core i7-4770 3.40GHZ.
- Memoria RAM: 8GB.
- Sistema Operativo: Windows 8.1.

La versión del lenguaje de Programación Python que se utiliza es la 2.7.

Es importante mencionar que para la experimentación se utilizan pesos fijos de 1 en la función de evaluación, tanto para la función de factibilidad como para la función heurística, de esta manera se pueden tener los valores reales de cada combinación de parámetros de los algoritmos. Una vez que se tenga el sistema, se puede modificar el peso de cada restricción de acuerdo a su importancia de la misma para que los algoritmos se enfoquen en satisfacer las restricciones con más colisiones, o en el caso de las restricciones débiles, se puede aplicar un decremento del peso por cada iteración.

4.2. Algoritmo Genético

Para obtener los parámetros de la experimentación, a continuación se muestran algunos ejemplos realizados para observar el comportamiento del Algoritmo Genético, la función de evaluación en éstas pruebas no cuenta con la función heurística debido a que éstas pruebas se realizaron antes de que se tuviera codificado la función de evaluación que se utiliza en la implementación final. El valor aproximado de la función heurística es de 1517, por lo que se muestran valores de función de evaluación menores a los valores de las pruebas finales, ésto mismo se contempla para los ejemplos de los demás algoritmos.

Primero, en el ejemplo 1 de la Figura 4.2 se puede observar que hay veces en las que la función de evaluación disminuye y puede volver a aumentar

en unas cuantas iteraciones siguientes, por lo que es necesario tener varias iteraciones para obtener la estabilidad del algoritmo. Para este ejemplo se utilizaron: 100 generaciones, una población de 50 individuos, un porcentaje de cruza de 60 %, una probabilidad de mutación de 50 %, un porcentaje de mutación de 20 % y un porcentaje de selección de 20 %. Para éste ejemplo, se tuvo una función mínima de 1729 con un tiempo de ejecución de 1925 segundos.

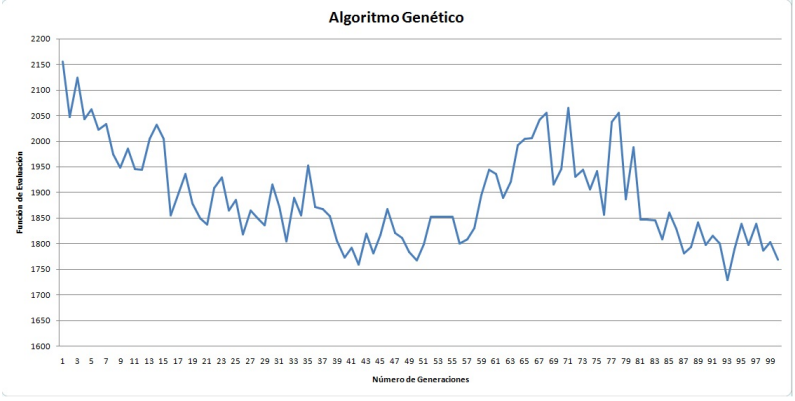


Figura 4.1: Ejemplo 1. Algoritmo Genético

En el siguiente ejemplo 2 de la Figura 4.2, la función de evaluación tiende a disminuir en unas pocas iteraciones, sin embargo, se mantiene con el mismo comportamiento en las siguientes iteraciones. También se puede observar que la función de evaluación tiende a aumentar en las últimas iteraciones. Los valores de los parámetros utilizados en este ejemplo son las siguientes: 300 generaciones, tamaño de población de 300 individuos, un porcentaje de cruza de 90 %, una probabilidad de mutación de 80 %, un porcentaje de mutación de 60 % y un porcentaje de selección de 50 %. Para éste ejemplo, se tuvo una función mínima de 1701 con un tiempo de ejecución de 437219 segundos.

Por último, en el ejemplo 3 de la Figura 4.3 se puede observar que la función de evaluación disminuye rápidamente, en este caso se utilizaron los siguientes parámetros: 150 generaciones, tamaño de población de 100 individuos, un porcentaje de cruza de 40 %, una probabilidad de mutación

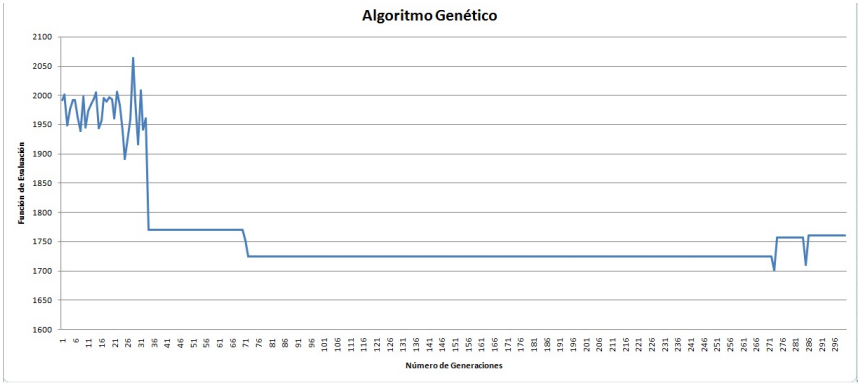


Figura 4.2: Ejemplo 2. Algoritmo Genético

de 50 %, un porcentaje de mutación de 10 % y un porcentaje de selección de 50 %. Para éste ejemplo, se tuvo una función mínima de 1424 con un tiempo de ejecución de 23470 segundos.

Tomando en cuenta los resultados anteriores, en algunos casos, conforme aumentan las generaciones el valor de la función de evaluación ya no disminuye de manera considerable, entonces es necesario utilizar otra condición de parada adicional. Esta nueva condición verifica que al no haber cambios considerables (positivos y negativos) en ciertas generaciones, se debe detener el algoritmo, ya que en las siguientes generaciones ya no disminuirá de manera considerable o se puede quedar estancado en la misma solución debido a que existen padres un tanto similares u otras condiciones no previstas.

Los parámetros propuestos para el algoritmo genético se muestran el la Tabla 4.1 y se escogen de tal manera que sean similares a lo que se describe en [1], pero con ligeras modificaciones entre cada parámetro para encontrar el que resuelva de manera adecuada el problema de horarios de la UTM, debido a que cada problema de horarios de una Universidad tiene diferente comportamiento. Además, los porcentajes de cruza y de mutación se eligieron a partir del comportamiento de los ejemplos mostrados anteriormente.

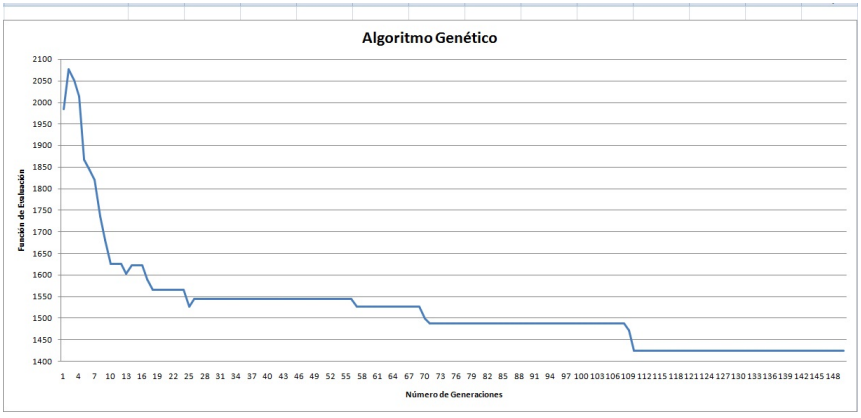


Figura 4.3: Ejemplo 3. Algoritmo Genético

NG	*TP*	*PCRUZA*	*PMUTA*	*PORMUTA*	*PORSELEC*
50	50	60	20	10	50
100	70	70	30	20	60
150	100	80	40	30	70

Tabla 4.1: Parámetros utilizados en el algoritmo Genético

Se utilizan las siguientes abreviaciones en los parámetros:

NG = Número de Generaciones.
TP = Tamaño de la población.
PCRUZA = Probabilidad de cruza.
PMUTA = Probabilidad de Mutación.
PORMUTA = Porcentaje de Mutación.
PORSELEC = Porcentaje de selección.

Utilizando estos parámetros se forman 729 combinaciones diferentes con las que se pueden obtener soluciones variadas y de las cuales alguna combinación debe ser mejor que las otras. En este algoritmo, la población inicial se genera de manera aleatoria y los padres son seleccionados de acuerdo al

valor de función de evaluación.

En la Tabla 4.2 se presentan los resultados del algoritmo Genético utilizando 50 ejecuciones de cada combinación de parámetros.

NG	*TP*	*CRUZA*	*MUTA*	*PMUTA*	*PSELEC*	*tMin*	*tProm*	*fEvMin*	*fEvProm*
50	50	80	30	30	50	**193.58**	662,06	3115	3221
50	50	60	30	20	50	377,78	**614.57**	2802	3032
50	70	60	30	10	50	874,1	1461,99	**2629**	2837
100	100	40	30	70	50	1463,23	2365,82	2677	**2752**

Tabla 4.2: Resultados del algoritmo Genético.

Se aprecia que el tiempo mínimo en que se ejecutó una combinación de parámetros del algoritmo es de 3 minutos y medio aproximadamente. El tiempo mínimo promedio en segundos de las ejecuciones es de 10 minutos aproximadamente y se obtiene con una combinación de parámetros diferente a la del tiempo mínimo.

Se puede observar que el valor de la función de evaluación es menor con respecto a los demás algoritmos. Sin embargo, se puede ver que el tiempo promedio mínimo con respecto a los demás algoritmos es más elevado (10 minutos con el menor valor de número de generaciones).

4.3. Algoritmo de Búsqueda Tabú

Para obtener los parámetros de la experimentación, se realizaron diferentes pruebas para observar el comportamiento del Algoritmo de Búsqueda Tabú. Primero, en el ejemplo 1 de la Figura 4.4 se puede observar que la función de evaluación disminuye de manera gradual, además no se ve una disminución drástica. Para este ejemplo se utilizaron: 150 generaciones, un tamaño de memoria de 10, un número de vecinos de 100 y un porcentaje de cambio de 30 %. Además, el algoritmo obtuvo una función mínima de 1907 con un tiempo de ejecución de 1137 segundos.

En el siguiente ejemplo 2 de la Figura 4.5, la función de evaluación se comporta de manera similar al ejemplo 1 a pesar de que ésta tenga más iteraciones. Para éste algoritmo en particular, el comportamiento de

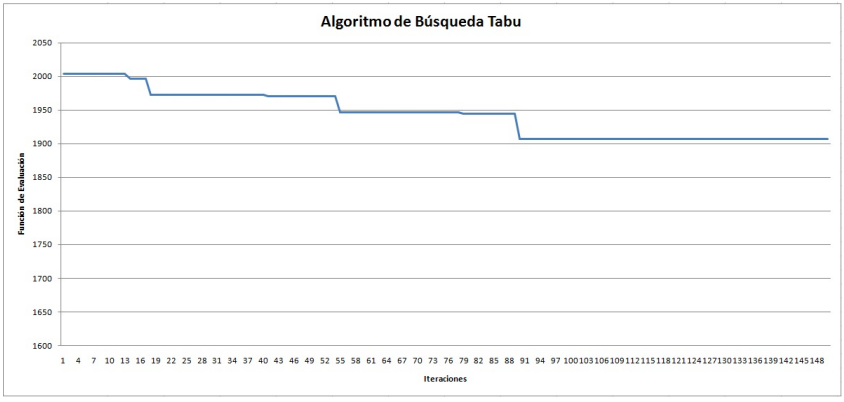


Figura 4.4: Ejemplo 1. Algoritmo de Búsqueda Tabú

los demás ejemplos es similar, por lo que no es necesario integrarlos. Los valores de los parámetros utilizados en este ejemplo son las siguientes: 150 generaciones, un tamaño de memoria de 20, un número de vecinos de 100 y un porcentaje de cambio de 20 %. Para éste ejemplo, se tuvo una función mínima de 1866 con un tiempo de ejecución de 1226 segundos.

En la experimentación con el algoritmo de búsqueda Tabú se utilizan los parámetros presentados en la Tabla 4.3. Éstos parámetros se obtienen a partir de la bibliografía y tomando en consideración las pruebas realizadas, se utiliza un tamaño de memoria tabú pequeño porque el algoritmo tiene un mejor desempeño de esta manera.

I	TM	NV	PM
100	10	50	5
150	25	75	10
200	50	90	20

Tabla 4.3: Parámetros utilizados en el algoritmo de Búsqueda Tabú

Se utilizan las siguientes abreviaciones en los parámetros:

I = Número de Iteraciones.

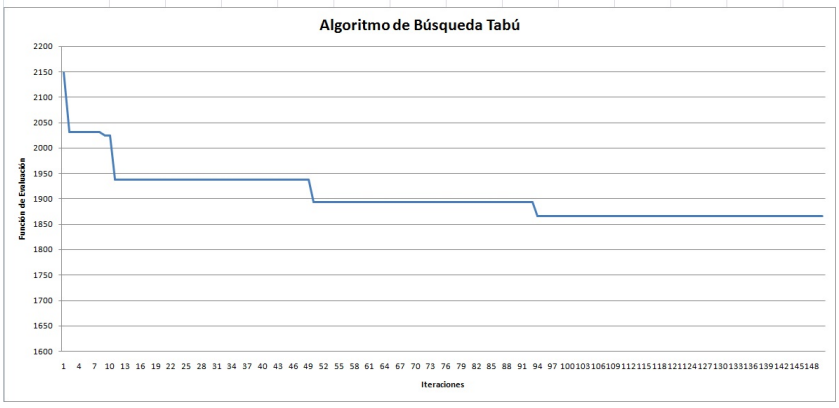


Figura 4.5: Ejemplo 2. Algoritmo de Búsqueda Tabú

TM = Tamaño de la memoria Tabú.
NV = Número de Vecinos.
PM = Porcentaje de Modificación de los vecinos.

En total se tienen 81 combinaciones diferentes de los parámetros, de las cuales uno debe minimizar la función de evaluación más que las demás.

En la Tabla 4.4 se presentan los resultados que se obtuvieron para el algoritmo de búsqueda Tabú, el tiempo mínimo y el tiempo promedio mínimo en segundos se obtuvieron con la misma combinación de parámetros, con 100 iteraciones, tamaño de memoria de 25, número de vecinos de 50 y porcentaje de modificación igual a 5; el tiempo mínimo que se obtuvo fue de 157.34 segundos, aumentando tan solo un poco en el tiempo promedio. También se observa que la función de evaluación mínima y la función de evaluación promedio mínima se obtuvieron con parámetros similares, para este resultado, se necesitaron 150 iteraciones, un tamaño de memoria de 10, 90 vecinos y un porcentaje de modificación de 20. Se puede notar que no fue necesario 200 iteraciones para alcanzar un buen resultado para este algoritmo. También el tamaño de memoria en este caso es pequeño, esta característica no es común en este algoritmo, por lo que se puede concluir que el algoritmo en este caso trabaja mejor con intensificación en vez de

diversificación de la búsqueda.

I	TM	NV	PM	$tMin$	$tProm$	$fEvMin$	$fEvProm$
100	25	50	5	**157.34**	**157.53**	3277	3331
150	10	90	20	430,16	431,20	**3022**	**3251**

Tabla 4.4: Resultados del algoritmo de Búsqueda Tabú.

4.4. Algoritmo de Recocido Simulado

Para obtener los parámetros utilizados, se utilizan valores similares a [11], además se realizaron algunas pruebas con diferentes parámetros para obtener un mejor acercamiento a los buenos parámetros para el problema de horarios de la UTM. En las Figuras 4.6 y 4.7 se muestran dos ejemplos del comportamiento de los parámetros para este algoritmo.

En el ejemplo 1 se muestra que a pesar de que se tienen varias iteraciones como parámetro, el algoritmo termina en tan solo 21 iteraciones, ésto debido a que se escogen mal los parámetros de temperatura. El algoritmo de recocido simulado utiliza los valores de temperatura en su procedimiento. Los parámetros utilizados en este ejemplo son los siguientes: 200 iteraciones, temperatura inicial de 20, temperatura mínima de 0.01, número de vecinos de 150, porcentaje de cambio de 10 % y una reducción de temperatura de 0.7. La función mínima que se obtuvo a través de este ejemplo fue de 1966 y tuvo un tiempo de ejecución mínima de 612 segundos.

En el caso del ejemplo 2 mostrado en la Figura 4.7, se puede observar que el algoritmo puede quedarse estancado con los mismos valores de evaluación, para eso es necesario probar con varias combinaciones y reiniciar el algoritmo si es posible. Los parámetros utilizados en este ejemplo son los siguientes: 150 iteraciones, temperatura inicial de 100, temperatura mínima de 0.0001, número de vecinos de 50, porcentaje de cambio de 40 % y una reducción de temperatura de 0.97. La función mínima que se obtuvo a través de este ejemplo fue de 1891 y tuvo un tiempo de ejecución mínima de 3572 segundos.

Los parámetros utilizados en la experimentación del recocido simulado se muestran en la Tabla 4.5 y en total se tiene una combinación de 729 diferentes valores de entrada para el algoritmo.

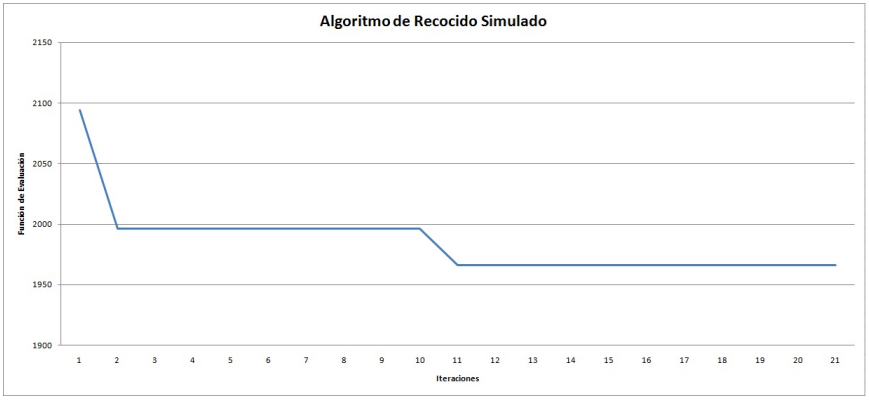


Figura 4.6: Ejemplo 1. Algoritmo de Recocido Simulado

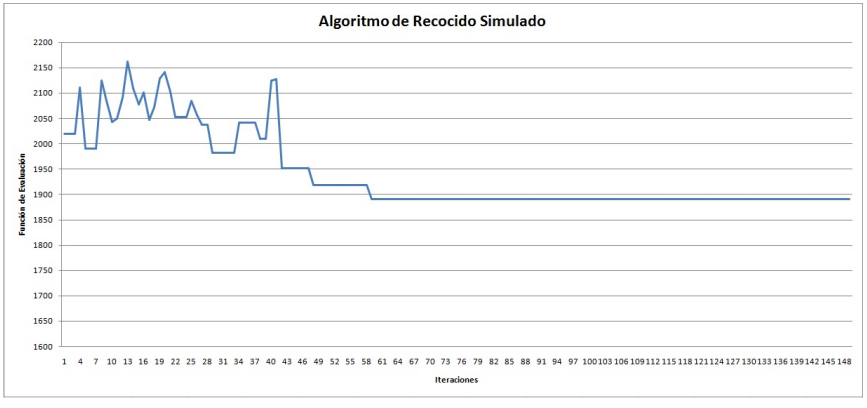


Figura 4.7: Ejemplo 2. Algoritmo de Recocido Simulado

I	TI	TM	NV	MOD	DT
100	25	0,00001	20	20	0,9
150	50	0,0001	30	40	0,92
200	100	0,001	50	50	0,97

Tabla 4.5: Parámetros utilizados en el algoritmo de Recocido Simulado

Se utilizan las siguientes abreviaciones en los parámetros:

I = Número de Iteraciones.
TI = Temperatura Inicial.
TM = Temperatura Mínima.
NV = Número de Vecinos.
MOD = Porcentaje de Modificación de la Temperatura.
DT = Decremento de la Temperatura.

El rango de temperatura que se propone en [11] es de [1,100], es por eso que en base a pruebas parciales y de acuerdo a la bibliografía se proponen las temperaturas iniciales que se presentan en la Tabla 4.5. La temperatura mínima se maneja un poco menor a lo que debería ser debido a que en algunos casos observados en los ejemplos presentados, la temperatura llega a alcanzar demasiado pronto la temperatura mínima esperada sin haberse encontrado una solución buena.

En la Tabla 4.6 se presentan los resultados del algoritmo de Recocido Simulado, en donde se obtienen de manera preliminar las ejecuciones del algoritmo con 20 ejecuciones de cada combinación de parámetros.
La función mínima en promedio de las diferentes combinaciones. El tiempo mínimo y el tiempo promedio mínimo se obtuvieron con la misma combinación de parámetros, además la función de evaluación mínima se obtuvo con dos combinaciones de parámetros diferentes y la función de evaluación promedio mínima se obtuvo con 150 iteraciones, temperatura inicial de 100, temperatura mínima de 0.0001, número de vecinos de 50, porcentaje de modificación de 40 y un decremento de temperatura de 0.97.

El tiempo del algoritmo de recocido simulado es tardado, 5 minutos

I	TI	TM	NV	MOD	DT	$tMin$	$tProm$	$fEvMin$	$fEvProm$
100	25	0,001	20	20	0,9	**259.99**	**260.65**	3318	3376
100	25	0,0001	30	50	0,97	806,36	807,67	**3111**	3335
150	100	0,001	50	20	0,92	927,06	928,95	**3111**	3294
150	100	0,0001	50	40	0,97	1682,65	1683,47	3160	**3276**

Tabla 4.6: Resultados del algoritmo de Recocido Simulado.

aproximadamente por ejecución de una combinación de parámetros en el mejor de los casos.

4.5. Algoritmo de Búsqueda Armónica

Para nuestro problema particular, se realizaron algunas pruebas para obtener los parámetros adecuados, sin embargo este algoritmo en particular no tiende a disminuir mucho en cuanto a función de evaluación. A continuación en el ejemplo de la Figura 4.8 se muestra que el algoritmo tiende a reducirse en pocas ocasiones durante su iteración. Los valores de parámetros utilizados en este ejemplo son los siguientes: 400 iteraciones, tamaño de Memoria de 50, Razón de exploración de 50 %, ancho de desplazamiento de 90 % y razón de ajuste de tono del 10 %. La función mínima que se obtuvo a través de este ejemplo es de 2034 y tuvo un tiempo de ejecución mínima de 3552 segundos.

En este caso, basándose en el ejemplo de la Figura 4.9 se puede decir que el algoritmo no disminuye cuando tiene valores de parámetros que no le favorecen. Los valores de parámetros utilizados en este ejemplo son los siguientes: 150 iteraciones, tamaño de Memoria de 10, Razón de exploración de 60 %, ancho de desplazamiento de 70 % y razón de ajuste de tono del 50 %. La función mínima que se obtuvo a través de este ejemplo es de 2118 y tuvo un tiempo de ejecución mínima de 274 segundos.

Los valores de los parámetros para realizar las pruebas con este algoritmo se escogieron de tal manera que sean parecidas a otros trabajos para resolver el mismo problema, tal como en [4], sin embargo en este artículo no se tienen tan buenos resultados por lo que se complementan los datos con [50].

Al igual que los otros algoritmos, en la Tabla 4.7 se presentan los

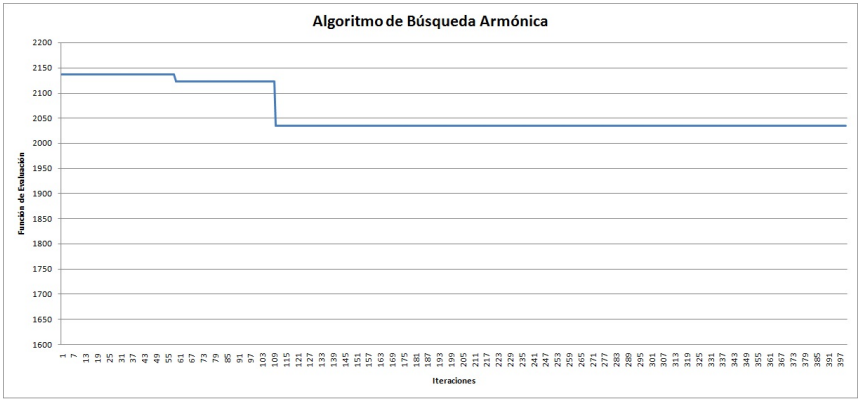


Figura 4.8: Ejemplo 1. Algoritmo de Búsqueda Armónica.

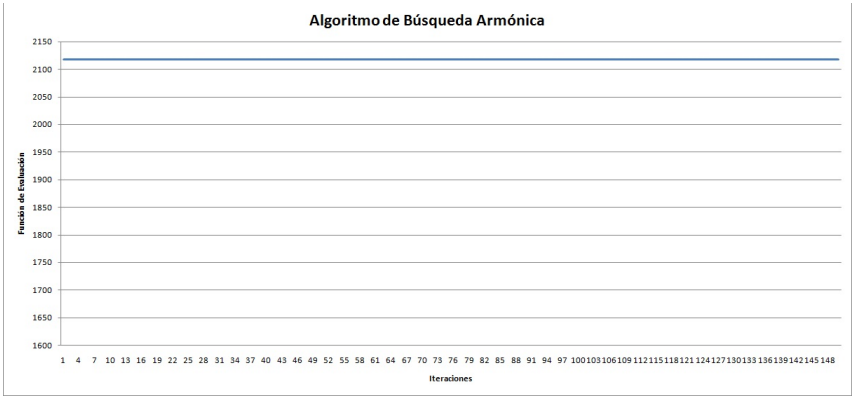


Figura 4.9: Ejemplo 2. Algoritmo de Búsqueda Armónica.

parámetros utilizados para la experimentación con el algoritmo de Búsqueda Armónica.

I	HMS	$HMCR$	BW	PAR
100	10	60	20	10
150	20	75	35	25
200	40	90	60	50

Tabla 4.7: Parámetros utilizados en el algoritmo de Búsqueda Armónica.

Se utilizan las siguientes abreviaciones en los parámetros:
I = Número de Iteraciones.
HMS = Tamaño de la Memoria Armónica.
HMCR = Razón de Exploración.
BW = Ancho de Desplazamiento.
PAR = Razón de Ajuste de Tono.

En total se tienen 243 combinaciones diferentes de parámetros de entrada del algoritmo armónica, de las cuales una combinación debe ser mejor que las demás.

Debido a que la memoria armónica guarda buenas soluciones, se especifican valores bajos para éste, de manera que se pueda sustituir a las soluciones no factibles en un periodo corto de tiempo.

A continuación, en la Tabla 4.8 se presentan los resultados del algoritmo de búsqueda armónica, en donde se obtienen de manera preliminar las ejecuciones del algoritmo con 20 ejecuciones de cada combinación de parámetros.

La combinación de parámetros del algoritmo para la cual se obtuvo el tiempo mínimo en segundos fue con 100 iteraciones, un tamaño de memoria armónica de 10, una razón de exploración de 75, un ancho de desplazamiento de 35 y con la razón de ajuste de tono de 50.
Para el tiempo promedio mínimo hubo cambios en la razón de exploración, el ancho de desplazamiento y en la razón de ajuste de tono.

La función de evaluación mínima se obtuvo con 100 iteraciones, de manera diferente para la función de evaluación promedio mínima se obtuvo

con 150 iteraciones.

I	HMS	$HMCR$	BW	PAR	$tMin$	$tProm$	$fEvMin$	$fEvProm$
100	10	75	35	50	**58.47**	59,77	3450	3580
100	10	90	60	25	59,17	**59.36**	3440	3615
100	40	75	35	25	222,44	223,71	**3273**	3479
150	20	60	60	25	166,42	167,68	3324	**3465**

Tabla 4.8: Resultados del algoritmo de Búsqueda Armónica.

En estos primeros resultados podemos observar que aunque no se observan resultados prometedores en la parte de valor de función de evaluación, este algoritmo puede hallar algunas buenas soluciones de manera rápida.

4.6. Comparativa y Decisión

Para mostrar el desempeño de cada uno de los algoritmos, es preciso obtener un promedio del valor de la función de evaluación inicial de las soluciones generadas. A partir de 1000 soluciones generadas aleatoriamente se calcula un promedio de **3916**, la cual se utiliza como referencia de comparación para los cuatro algoritmos meta-heurísticos y posteriormente para la hiper-heurística.

En las siguientes gráficas se muestran los resultados obtenidos de la experimentación de los algoritmos meta-heurísticos. Primero, en la gráfica de la Figura 4.10 se muestra el tiempo de ejecución promedio mínimo de los cuatro algoritmos seleccionados. Se puede ver en la gráfica que el Algoritmo Genético tarda en ejecutarse en el mejor de los casos 10 minutos aproximadamente, éste valor puede aumentar considerablemente en el peor de los casos, o mas bien dicho cuando los valores de las variables de entrada sean las mas elevadas (número de generaciones, tamaño de población, etc.); lo mismo sucede con los demás algoritmos, sin embargo, la gráfica muestra la velocidad de ejecución de los algoritmos y no necesariamente el algoritmo más veloz tiene el mejor desempeño en cuanto a valor de función de evaluación como se muestra en la gráfica de la Figura 4.11, en donde se puede observar que el algoritmo genético es el que tiene un mejor desempeño en cuánto a los tres algoritmos Meta-Heurísticos restantes.

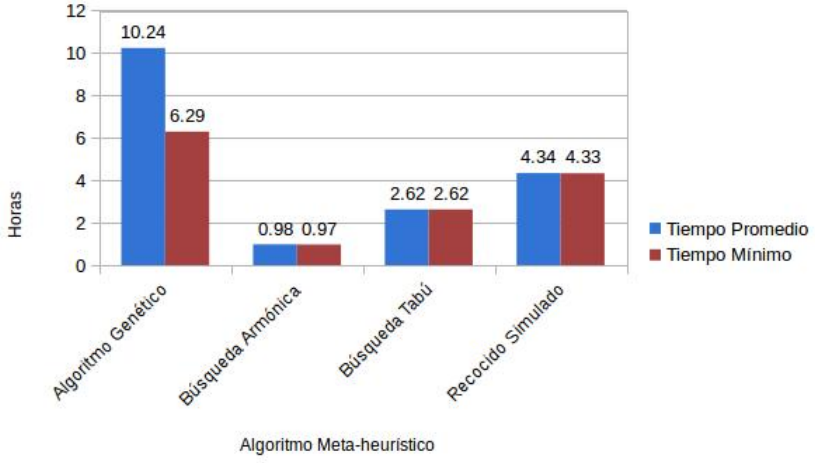


Figura 4.10: Gráfica de comparación en función al tiempo de ejecución de los cuatro algoritmos implementados.

A partir de la comparación de los cuatro algoritmos meta-heurísticos se define el algoritmo base para la hiper-heurística, la cual es, el Algoritmo de Búsqueda Tabú, por el hecho de que es el algoritmo que tiene un buen tiempo de ejecución (sólo por debajo del algoritmo de búsqueda armónica) y porque su valor de función de evaluación promedio mínimo está en segundo lugar, por debajo del algoritmo genético. Para este trabajo de tesis no se considera el Algoritmo Genético porque es muy costoso en cuánto a experimentación tomando en cuenta los valores reales del problema (se consideran 50 ejecuciones de cada una de las 729 combinaciones de parámetros), a pesar de que tiene el mejor valor de función de evaluación promedio mínimo.

En el siguiente capítulo se detallan el uso de los algoritmos propuestos en el algoritmo hiper-heurístico, así como el procedimiento para su implementación y las modificaciones que se le realizaron a la función de evaluación y a los algoritmos a utilizar.

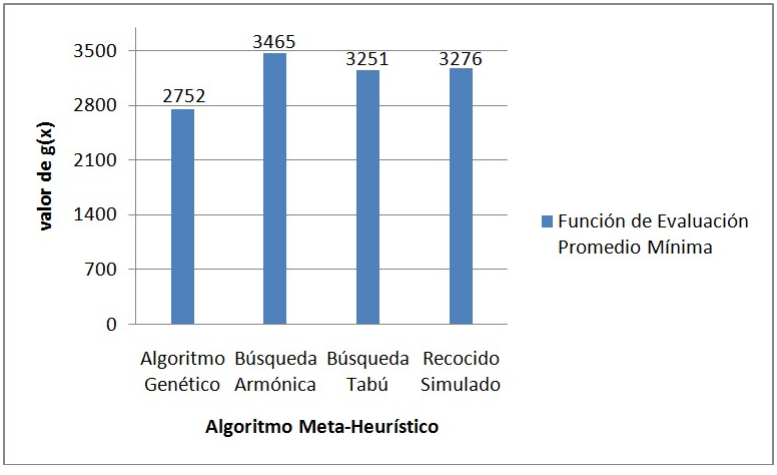


Figura 4.11: Gráfica de comparación con respecto a la función de evaluación de los cuatro algoritmos implementados.

Capítulo 5

Diseño de Hiper-Heurística

En este capítulo se describe el diseño e implementación de la Hiper-heurística basada en Búsqueda Tabú, se toma en consideración los parámetros y el mejor algoritmo en cuánto a función de evaluación de la comparativa de los cuatro algoritmos Meta-Heurísticos. Además se considera una función nueva para determinar el camino a seguir durante las iteraciones del algoritmo a implementar.

En secciones posteriores se describen las meta-heurísticas y heurísticas locales utilizadas y el motivo de porqué se utilizan en ciertas partes del algoritmo.

5.1. Modificaciones a la Función de Evaluación

En las meta-heurísticas la función de evaluación se utiliza para evaluar la calidad de una solución, permitiendo la búsqueda de una solución que cumpla con el mayor número de requerimientos solicitados por la institución educativa.

Así mismo, en la hiper-heurística se pretende utilizar una función de evaluación similar a la descrita en la metodología de las meta-heurísticas. Pero en esta ocasión se agrega una **función de selección** *que permita evaluar cada una de las restricciones fuertes y débiles del problema*, al mismo tiempo debe obtener las posiciones (del vector utilizado como solución) para cada restricción que generen las colisiones.

En cada iteración del algoritmo Hiper-Heurístico, se utiliza la función de selección para obtener el problema a solucionar en la siguiente iteración y con la función de evaluación se hace seguimiento de la calidad de la solución para cada algoritmo de búsqueda local aplicado por la Hiper-Heurística. La función de Selección se define como sigue:

$$e(x, y) = max(costo_n(x) - costo_n(y)) \tag{5.1}$$

donde
$costo_n(x)$ = costo de la n-esima restricción (fuerte y débil),
$costo_n(y)$ = costo de la n-esima restricción anterior (fuerte y débil)

La modificación a la función de evaluación que se presenta en la presente sección permite decidir cuáles serán las heurísticas locales o meta-heurísticas a aplicar de acuerdo a sus capacidades de mejora de alguna o algunas de las restricciones del problema.

5.2. Heurísticas Locales

A pesar de que las meta-heurísticas tienden a ser costosas en tiempo de ejecución, durante el proceso de búsqueda de una Hiper-heurística existen búsquedas locales que necesitan menos tiempo de procesamiento, ya que se dedican a resolver pocas colisiones. Es así como se plantea complementar la hiper-heurística con algunas heurísticas locales para disminuir el tiempo de búsqueda global.

Las heurísticas locales que se implementan son el algoritmo de Descenso de Colina y Descenso de Colina con escalada Simple. El algoritmo de Descenso de Colina se utiliza en casos donde una restricción no tenga tantas colisiones, ya que si se utiliza en una restricción donde hay varias colisiones, no se podrá asegurar que se encuentre una mejor solución a la que se tiene actualmente y el algoritmo podría quedarse estancado. Sin embargo, si se aplica en restricciones que no tengan tantas colisiones y no mejora la solución, no afectará en gran medida a la factibilidad global.

Para las restricciones que tengan casi todas sus colisiones resueltas, se utiliza el algoritmo de Descenso de Colina con escalada simple, para no afectar a la función de evaluación global, además, no es necesario implementar

un algoritmo que consuma mucho tiempo debido a que las colisiones son mínimas y tan sólo aumentaría el tiempo de proceso de la Hiper-Heurística.

5.2.1. Recocido Simulado y Búsqueda Armónica

El algoritmo de Recocido Simulado y de Búsqueda Armónica también se utilizan como algoritmos de búsqueda local para la Hiper-Heurística, debido a que en situaciones donde hay varias colisiones es necesario un algoritmo que mejore la solución actual y por ese motivo se escogen éstos dos algoritmos, tomando en cuenta los mejores parámetros encontrados en la comparativa para su implementación en la hiper-heurística.

El algoritmo de Recocido Simulado se utiliza para solucionar la restricción con más colisiones, debido a que es un tanto tardado solamente se utiliza en dicha restricción, pero se contempla que puede mejorar la solución global al encontrar una solución que disminuye las colisiones de la restricción que esté procesando el algoritmo.

El algoritmo de Búsqueda Armónica se utiliza para la segunda restricción con más colisiones, se escoge este algoritmo porque es un poco más rápido que el algoritmo de recocido simulado.

5.3. Hiper-heurísticas

La hipocorística que se utiliza para resolver el problema de calendarización se define y se implementa en las siguientes secciones. Previamente en el capítulo 2 se describen algunas definiciones importantes que se utilizan en este capítulo.

5.3.1. Descripción

Para resolver el problema de horarios de la UTM por medio de una hiper-heurística, es necesario obtener y aplicar conocimiento en cada paso de la hiper-heurística para atacar los problemas específicos que limitan la búsqueda hacia una mejor solución. Como conocimiento nos referimos a identificar los diferentes problemas que impiden disminuir la función de

factibilidad de una solución durante las iteraciones de los algoritmos utilizados.

Los diferentes problemas en una solución se presentan por el incumplimiento de las restricciones fuertes y blandas, lo que puede llevar el problema a una dirección donde la solución se estanca o donde solamente se encuentran soluciones similares o con mayor función de evaluación en los vecindarios. Entonces, para poder seguir minimizando el valor de la función de evaluación principal, en la hiper-heurística basada en Búsqueda Tabú se obtiene información de cada restricción para comparar el rendimiento de cada iteración que muestre el grado de influencia de cada restricción en la solución con respecto a la función de factibilidad.

Primero, como paso previo a la implementación de la hiper-heurística, se realiza un seguimiento de lo que sucede en cada iteración utilizando una función de selección, esta función calcula el número de colisiones de cada restricción. Una vez que se calcula el número de colisiones de cada restricción, se comparan con el número de colisiones de las restricciones de una solución anterior, luego, se obtiene el máximo de las diferencias entre cada elemento así como se muestra en la Ecuación 5.1. Para obtener el máximo de la primera iteración se genera una lista del promedio de colisiones en cada restricción utilizando 1000 soluciones aleatorias y destacando las colisiones de cada restricción como se presenta en la Tabla 5.1.

factibilidad 1	6
factibilidad 2	15
factibilidad 3	231
factibilidad 4	1577
factibilidad 5	464
factibilidad 6	9
factibilidad 8	5
factibilidad 9	87
heurística 1	691
heurística 2	820

Figura 5.1: Colisiones en una solución.

Las cifras que se presentan en la Tabla 5.1 son el número de colisiones en promedio con las que inicia una solución en el algoritmo. Se puede

observar que estos valores no son nada favorables para la solución y afectan de manera directa el valor de la función de evaluación.

Realizando varias pruebas con los algoritmos meta-heurísticos implementados, se encuentra que las colisiones de cada restricción en una solución aumentan o disminuyen de diferente manera en el proceso de búsqueda de la solución final. Por ejemplo, si las colisiones de la restricción número 4 disminuyen en una iteración, es muy probable que aumenten en la siguiente iteración debido a que en cada iteración se atacan diferentes restricciones y ésto conlleva a la modificación del resultado obtenido en cada iteración. Por tal motivo, las colisiones de las restricciones estarán en constante cambio, pero de siempre minimizándose de manera global debido al algoritmo de búsqueda Tabú base que controla la aceptación de las soluciones. Así, para obtener una solución al problema, se realiza un seguimiento de las colisiones que presentan las restricciones de la solución, ésto, en cada iteración del algoritmo.

Tomando en cuenta las colisiones de cada restricción en una solución, se pretende que cada restricción pueda disminuir sus colisiones de manera independiente (sin afectar demasiado a las otras) y más eficiente. Para realizar este proceso, es necesario **evaluar la solución** con la función de selección presentada en la ecuación 5.1, de manera que se pueda obtener el **problema de la solución**, éste problema va a ser procesado por el algoritmo Hiper-heurístico para encontrar una **nueva solución** con un valor de función de evaluación menor, en el mejor de los casos. Éste proceso va a permitir que mientras más se avance en la búsqueda, se van a encontrar soluciones con una función de factibilidad menor por el hecho de que se atacan los problemas de manera separada.

Se puede decir que el *problema de la solución* es aquella restricción que al hacerle alguna modificación, en vez de disminuir sus colisiones , aumenta, o en algunos casos, es el que menos disminuye sus colisiones en una iteración. Hay ciertas restricciones que tienen un número alto de colisiones, éstas se consideran restricciones que deben solucionarse con algoritmos de más alto nivel, por lo que a éstas se les ataca con las meta-heurísticas. A las restricciones que normalmente tienen pocas colisiones en promedio, se les ataca con los algoritmos heurísticos.

A continuación, en la Figura 5.2 se presenta el modelo de Hiper-Heurística que se propone para resolver el problema de Horarios de la UTM.

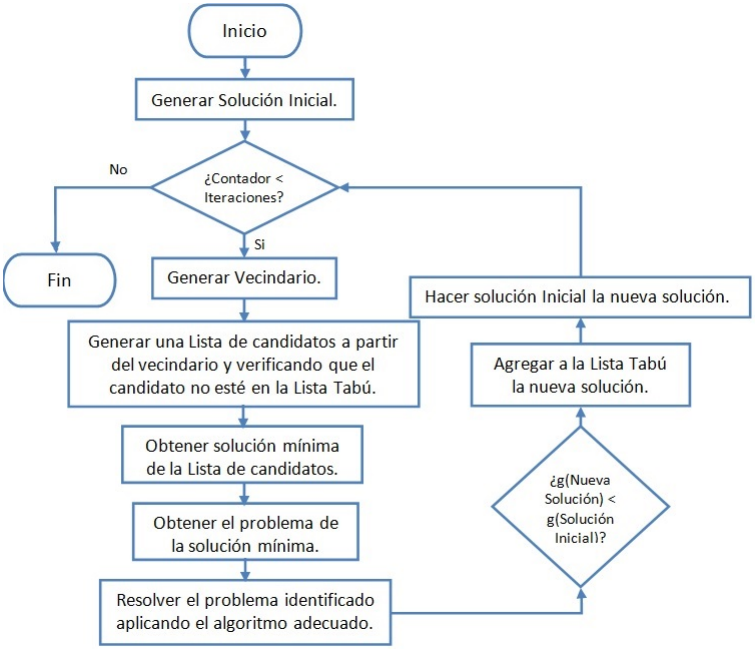


Figura 5.2: Diagrama de la propuesta de la hiper-heurística basado en Búsqueda Tabú.

Las restricciones con más colisiones identificados durante el análisis de la solución son las siguientes:

1.- *Espacio*: La restricción número 4 (Dos o más profesores o grupos no pueden estar en el mismo lugar, a la misma hora, el mismo día, a menos que sea una multiasignación) es la que genera más colisiones. Esto significa que hay varios grupos y profesores que se asignan en el mismo espacio.

Se intentará resolver aplicando como búsqueda local un algoritmo de Recocido Simulado de manera que se puedan generar vecinos con características diferentes en las colisiones. Se elige este algoritmo porque la restricción 4 es la que tiene más colisiones y además engloba

tanto colisiones de profesores como de grupos, lo cual la vuelve más compleja de resolver. Hay que tener en cuenta que, a pesar de que existe un gran número de espacios, la mayor parte de las asignaciones se restringen a un número mucho más reducido de los mismos debido a las necesidades de equipamiento de los espacios reportadas por cada profesor para impartir dicha asignatura.

2.- *Grupo*: La restricción número 5 (un grupo no puede tener asignada otra clase, ya sea en el mismo o diferente lugar a la misma hora, el mismo día, a menos que sean clases de diferentes especialidades de ese grupo) tiene menos colisiones que los problemas de Espacio y se refiere a que los grupos están distribuidos en espacios que ya contienen alguna asignación. A este problema se le aplica un algoritmo de Búsqueda Armónica debido a que éste algoritmo no es tan tardado y puede encontrar nuevas soluciones en poco tiempo. Además de que entre la restricción con más colisiones y la segunda restricción con más colisiones hay aproximadamente 300 % de diferencia, por lo que si este algoritmo ya no encuentra buenas soluciones, dependerá del primer algoritmo disminuir más la factibilidad global.

3.- *Profesor*: Éste problema pertenece a las restricciones fuertes, específicamente la número 3. Se refiere a que un profesor no puede estar en dos lugares al mismo tiempo impartiendo clases.

A este problema se le aplica la heurística local de Descenso de Colina, debido a que tiene menos colisiones que las otras dos mencionadas en el punto 1 y 2. Es decir, un espacio de búsqueda menor. Además se considera una iteración de búsqueda de 200 para que se tengan más resultados en la comparativa de la búsqueda.

4.- Para las *restricciones fuertes restantes* se considera utilizar el algoritmo de Descenso de Colina con Escalada Simple, debido a que no representan restricciones con varias colisiones, de tal manera que aplicando este algoritmo se pretende minimizar el tiempo de ejecución de la Hiper-Heurística. Para éstas restricciones, el número de Iteraciones varía dependiendo de las colisiones en cada restricción.

5.- Las restricciones *Heurísticas 1 y 2* se resuelven con Descenso de Colina, modificando el número de iteraciones para cada una, el primero

de 200 y el segundo de 400 iteraciones. Se utiliza este algoritmo ya que la función heurística no afecta a la solución final, sin embargo se debe tratar de reducir de igual manera, por tal motivo se utilizan varias iteraciones en su resolución para hacer una búsqueda más exhaustiva. La primera heurística tiene menos colisiones que la primera, por lo que se utilizan menos iteraciones y además se consideran éstos valores para no incrementar tanto el total del tiempo de ejecución de la Hiper-Heurística.

5.3.2. Implementación

Una solución consta de diferentes asignaciones y cada asignación se representa por varios enteros que representan la información de la asignación, tales como el Espacio, la Hora y/o el día de la asignación.

Para encontrar el problema a solucionar en la hiper-heurística, se recorre la solución completa buscando los inconvenientes sujetos a las restricciones duras y blandas, si en alguna asignación se encuentra una restricción incumplida, ésta se agrega a una lista para ser procesada por el algoritmo hiper-heurístico. Así para todas las restricciones, se utilizan diferentes listas para cada una de las restricciones que determinan las posiciones de colisiones en una solución. Todo este proceso se realiza durante el cálculo de la función de Selección para no realizar la tarea de buscar las posiciones de las colisiones y el número de colisiones de cada restricción en funciones diferentes.

Inicialmente se obtuvieron las asignaciones como los problemas a resolver, pero ya que con éste método se presentaron problemas para reducir la función de factibilidad, se optó por elegir los datos de las asignaciones (espacio, hora y día) como los problemas de las colisiones en una solución. Éstos problemas encontrados se explican en mayor detalle en el capítulo de experimentación, en la parte de hiper-heurística.

Para resolver el problema de mejor manera, se utilizan diferentes tipos de datos que se identifican en una solución dada su composición o naturaleza. Éstos datos son:

- *Datos Estáticos*: Son los datos de una solución que no se pueden modificar debido a que solamente cuentan con una posibilidad de elección. Por ejemplo, una clase debe ser impartida en un espacio

Figura 5.3: Búsqueda del problema en una solución.

específico, debido a que es el único espacio adecuado que cuenta con los recursos necesarios para que se lleve a cabo dicha clase. Así se tiene que en los Bounds, para esa asignación en específico, solamente existirá un id de espacio para poder ser elegido.

- *Datos Prioritarios*: Éstos datos cuentan con un tamaño reducido de posibilidades a elegir dentro de los Bounds, al igual que los datos estáticos, pueden ser datos de espacios, horas y/o días, pero en este caso debe ser de tamaño mayor a 1 y menor o igual a 3. Al ser prioritarios, primero se buscan soluciones que se adecuen a éstos datos. De esta manera, al encontrar una solución que cumpla con los requerimientos de estos datos, se pueden bloquear éstos datos en la elección para que no se vuelvan a elegir para generar una solución nueva, así solamente quedan los datos de No colisiones para formar nuevas soluciones.

- *Datos de No Colisiones*: Son los datos que tienen una posibilidad de elección en los Bounds mayor a 3, éstos datos se pueden elegir de manera aleatoria ya que contienen más posibilidades que las anteriores. Las diferentes técnicas o métodos de elección se pueden aplicar a éstos datos para reducir y encontrar una buena solución.

Para la implementación del Algoritmo Hiper-Heurístico basado en Búsqueda Tabú, los valores de los parámetros que se utilizan en este algoritmo son los que se obtuvieron como el mejor resultado del algoritmo de Búsqueda Tabú de la Meta-Heurística.

Para implementar el algoritmo de Recocido Simulado como búsqueda

local dentro de la Hiper-Heurística, se utilizan los valores de los parámetros con los que se obtuvo el mejor resultado durante la ejecución de los experimentos del algoritmo de Recocido Simulado de la Meta-Heurística. Es necesario utilizar los mejores parámetros obtenidos en las experimentaciones de las meta-heurísticas para que de esta manera se contemple que se obtendrán soluciones que mejoren el resultado encontrado con las meta-heurísticas.

A continuación, se listan las diferentes versiones que se realizaron al algoritmo de Búsqueda Tabú durante el proceso de implementación del algoritmo Hiper-Heurístico, debido a que al implementar el algoritmo y realizar las pruebas correspondientes, inicialmente no se pudo obtener una solución con un valor de evaluación que mejorara de manera potencial los resultados de las meta-heurísticas presentadas.

1.- Primero se construyó el algoritmo utilizando una evaluación directa del problema, la cual consiste en verificar el máximo de colisiones de las restricciones en una solución. Ésto al principio toma en cuenta la restricción número 4 como máximo ya que es la que provoca más colisiones y así se pueden resolver las colisiones de ésta restricción de manera exhaustiva.

El problema que se tuvo con éste método fue que en varias ocasiones las colisiones de la restricción 4 no se reducía hasta llegar a una cantidad igual o menor que las colisiones de la restricción que le antecede en número de colisiones. Por ejemplo, si la restricción 4 tuviera 500 colisiones, y al momento de resolver estas colisiones tan solo se reducen a 450 sin rebasar esta cantidad, pero las colisiones de la restricción 5 es de 400, entonces siempre se tratará de resolver la restricción 4, cuando el algoritmo debe tratar de resolver las colisiones de todas las restricciones debido a que muchas veces una restricción no puede reducirse debido a la solución inicial que se genera.

2.- Dada la forma de evaluación mencionada en el punto 1, se realizó un cambio en la búsqueda de alternativas para reducir el número de colisiones de la restricción número 4, ya que ésta es la que tiene el mayor número de colisiones. Uno de las estrategias que se implementa

es: si ya no se pueden reducir las colisiones modificando los espacios de clases, se modifican las horas de clase.

También, aunque no se hayan reducido las colisiones de ésta restricción, se buscó reducir las demás colisiones utilizando un porcentaje de cambio, el cual toma de manera aleatoria la restricción a reducir.

Éstos métodos no influyeron demasiado en la reducción de la función de factibilidad de la solución.

3.- En este punto se implementó la selección del problema como se describe en la Ecuación 5.1. El primer problema notable que se obtuvo en iteraciones avanzadas fue que la solución se quedaba en un punto donde su función de evaluación no se podía reducir más.

4.- En este punto se realizaron pruebas de aceptación de la solución generada como mejor solución ya que en cada iteración se mejora una parte del problema. Así, se puede decir, que la solución va a converger en algún punto. Éste método no funcionó, debido a que las pruebas se realizan con los mismos parámetros de la mejor solución de la búsqueda Tabú y dado que solamente tiene 150 iteraciones, no alcanza a reducir el problema, siendo necesarios muchas más iteraciones. En 150 iteraciones con este método, la factibilidad de la solución se reduce de manera muy escasa.

5.- Dado el problema anterior, se tomó en cuenta el algoritmo de búsqueda Tabú original con su variante de *oscilación estratégica*. Así se implementó que a cada 5 % del total de iteraciones en las que la factibilidad de la mejor solución no presente algún cambio, se debe tomar en cuenta la siguiente solución que se genere aunque no sea menor a la mejor solución actual. Con este método se pretende salir del óptimo local para verificar otras soluciones.

6.- Para solucionar problemas en donde es posible modificar tanto espacios, horas o días, se utiliza una condición de probabilidad que elige el tipo de datos a modificar. En este punto, los problemas que se evalúan en la solución generan un vector que contiene el número de asignación que contiene la colisión y el tipo de colisión.

7.- Posteriormente se probó dejar fijas algunas asignaciones, modificando las siguientes de manera que no se elijan las que ya están fijas. El problema que se obtuvo es que hay vectores de tamaño 1 que representan tan solo una posibilidad de escoger un dato para una asignación (espacio, día u hora). Es así como es imposible eliminar ciertas colisiones dado que no se tiene ninguna opción para el cambio de los valores en el vector solución.

8.- Finalmente, se modificó la selección de los problemas en una solución, de tal manera que aparte de encontrar el problema a resolver, debe generar un vector que contenga las posiciones exactas de cada problema, éste vector muestra la posición de Espacio, Hora y/o día que tiene la colisión en lugar de considerar las posiciones de una asignación.

Contemplando los problemas que se tuvieron durante las pruebas en los cambios que se realizaron, se llegó a la implementación que se describe en el diagrama de la Figura 5.2. Al generar los vecinos de la solución inicial, se descartan los datos fijos, para que el porcentaje de modificación no se reduzca. Los datos prioritarios son procesados de tal manera que al encontrarse una solución que tenga todos los datos prioritarios asignados sin generar colisiones, éstas se conviertan en datos fijos y también sean descartadas del vecindario. En cambio los datos de No colisiones siempre se tendrán presente al realizar algún cambio en la solución.

En la Universidad Tecnológica de la Mixteca, el total de asignaciones que se deben realizar para generar la solución son 1077 datos (número de bounds), tomando en cuenta espacios, días y horas como los datos a elegir, a partir de esta cifra se tienen 310 datos estáticos que representan el 28 % del total, así mismo se tiene 45 datos prioritarios, que representa el 4.1 % del total de bounds. Los datos restantes pertenecen al grupo de no colisiones, que al menos 4 opciones para elegir un elemento de una asignación.

En la implementación del algoritmo Hiper-Heurístico se toman en cuenta los algoritmos Meta-Heurísticos Búsqueda Tabú, Recocido Simulado y Búsqueda Armónica, también se utiliza el algoritmo Heurístico Descenso de Colina y Descenso de Colina con Escalada Simple tal y como se menciona al final de la sección de descripción en este capítulo.

Capítulo 6

Discusión y Resultados

Cada medio año se le dedica bastante tiempo al proceso de creación de horarios (generalmente de dos a tres semanas), debido a que se realiza de manera manual. Es necesario automatizar éste proceso o al menos tener un sistema de apoyo con la que se pueda reducir el tiempo que se le dedica a esta actividad. Normalmente el problema, aún con la tecnología actual, es difícil de resolver debido a los recursos que son necesarios para encontrar una solución óptima. Tal como se menciona en el capítulo 3, se pueden generar $7{,}8 \times 10^{1057}$ soluciones posibles, por lo que es imposible buscar solución por solución las que no tengan colisiones de ningún tipo. Esta cifra es tan alta debido al tamaño de la Universidad Tecnológica de la Mixteca, ya que se necesitan asignar aproximadamente 200 profesores en 298 espacios para que impartan 246 materias a determinados grupos y tomando en cuenta que los espacios y profesores tienen restricciones.

6.0.1. Resultados

Después de llevar a cabo una considerable cantidad de pruebas con las cuatro meta-heurísticas comparadas y diseñado una hiper-heurística, en base a esas pruebas se obtuvieron dos tipos de resultados para resolver una instancia del problema de calendarización de horarios de la UTM. El primer tipo de Resultado es a partir del total de Asignaciones, que como se mencionó anteriormente, una asignación está compuesta por seis elementos, las cuales son el Profesor, Materia, Grupo, Espacio, Día(s) y Hora(s). El

segundo tipo de Resultado que se presenta está relacionado con el número total de colisiones para todas las asignaciones, misma que se calcula con la función de evaluación, en este caso, cada asignación puede tener varias colisiones que pertenecen a diferentes restricciones, aumentando el total de colisiones en la función de evaluación.

Resultados en base a Asignaciones

Las pruebas realizadas en este trabajo de Tesis corresponden al periodo escolar 2017-B de la UTM, en donde se tienen 440 asignaciones, de estas asignaciones se tienen los resultados de la Figura 6.1 con los diferentes algoritmos utilizados.

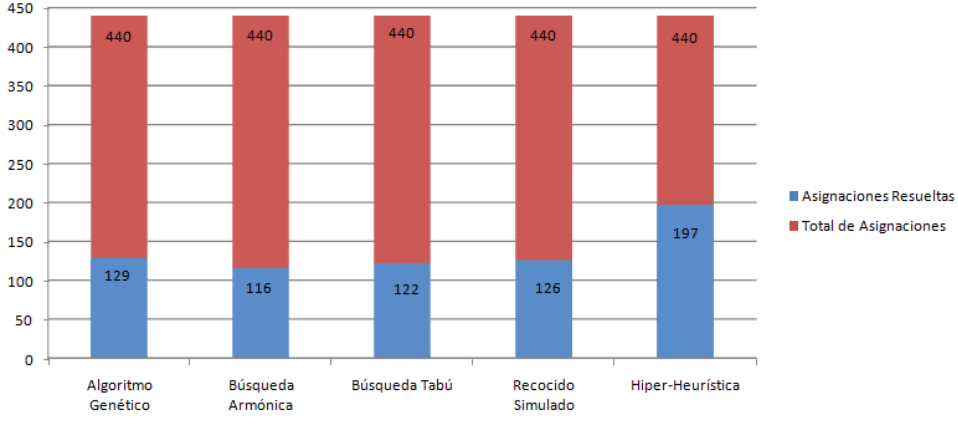


Figura 6.1: Número de asignaciones resueltas por los diferentes algoritmos utilizados.

Se puede observar que en todos los algoritmos utilizados y con los parámetros propuestos, el porcentaje de asignaciones resueltas es menos del 50 %, esto es debido a que en una asignación pueden existir colisiones de diferente tipo, relacionadas a las restricciones. Así, se toma en cuenta que si en una asignación existe al menos una colisión, se considera como no resuelta. Por ejemplo, en la Figura 6.2 se ve que hay 9 colisiones en un espacio, lo que hace que la asignación que representa el conjunto de datos

para esta asignación se considera no resuelta. Si en lugar del Espacio, se tuviera una colisión de horas repetidas para esta asignación, la asignación se consideraría de igual manera no resuelta. Lo mismo aplica para las demás asignaciones, se verifica que no tengan ninguna colisión de cualquier tipo.

Figura 6.2: Colisiones en un Espacio.

Resultados en base a Función de Evaluación

El algoritmo Hiper-Heurístico basado en Búsqueda Tabú descrito en el capítulo anterior obtuvo buenos resultados (29.7 % de reducción con respecto al mejor resultado de las Meta-heurísticas implementadas). Ésto se puede ver en la Figura 6.3 en donde se pueden observar el número de colisiones mínimo y promedio que se obtiene al aplicar la función de evaluación descrita en el capítulo 3 a un conjunto de soluciones iniciales generadas aleatoriamente en cada algoritmo, así se obtiene la mejor solución de cada una de las meta-heurísticas después de ciertas ejecuciones con sus mejores parámetros.

En esta gráfica se muestran el promedio y el mínimo del valor de función de evaluación que generan los algoritmos. El promedio se calcula almacenando todos los valores mínimos de función de evaluación que se obtienen en cada uno de las 50 ejecuciones de los algoritmos y se calcula el promedio de las mismas. El valor mínimo es el menor de todos los valores almacenados anteriormente.

Si se toman en cuenta *10000 soluciones* generadas de manera aleatoria, para posteriormente evaluarlas y obtener un *promedio de 3916* en su fun-

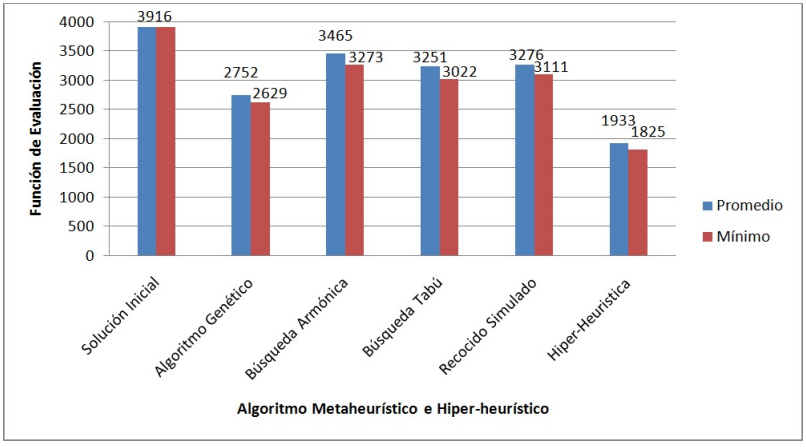


Figura 6.3: Resultado final de la Función de Evaluación para cada uno de los algoritmos.

ción de evaluación, esta cifra se utiliza como factor de comparación para verificar el desempeño de los algoritmos implementados. Es así como se tienen los siguientes resultados:

- *Algoritmo de Búsqueda Tabú:* tiene un valor de función de evaluación de **3251**, lo cual representa el **16.9 %** de reducción con respecto al promedio.
- *Algoritmo de Búsqueda Armónica:* tiene un valor de función de evaluación de **3465**, la cual representa una reducción de **11.5 %**.
- *Algoritmo de Recocido Simulado:* tiene un valor de función de evaluación de **3276**, lo que representa una reducción de **16.3 %**.
- *Algoritmo Genético:* tiene un valor de función de evaluación de **2752**, que representa un **29.7 %** de reducción con respecto al promedio.
- *Algoritmo Hiper-Heurístico:* tiene un valor de **1934**, la cual representa una reducción de **50.6 %**.

En las siguientes Imágenes se pueden apreciar los resultados obtenidos con los diferentes algoritmos.

La Figura 6.4 muestra los elementos importantes que se obtienen a partir de la ejecución de los diferentes algoritmos en el sistema de horarios, éstos elementos son:

1.- El elemento evaluado (Espacio, Grupo o Profesor) y que tiene diferentes asignaciones que deben cumplir con algunas restricciones.

2.- Total de colisiones en cada tipo de elemento evaluado, este número representa las colisiones sin resolver en el algoritmo, dependiendo del elemento evaluado.

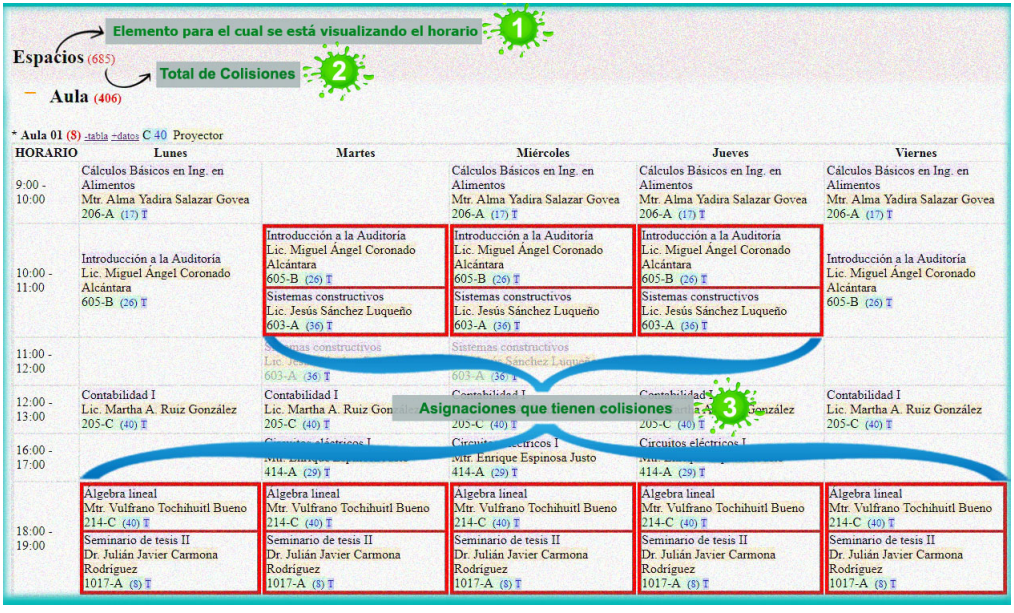
3.- Las asignaciones que tienen colisiones son los elementos de una asignación específica que no se pudieron resolver a través del algoritmo utilizado.

En la Figura 6.4 se puede ver que hay 8 colisiones para esa asignación, en donde se asignaron dos profesores en un mismo espacio, a la misma hora, el mismo día. Cabe mencionar que aunque realmente solo son dos asignaciones las que colisionan, se suman colisiones por cada día en el que se presentan.

En la Figura 6.5 se aprecia el número de colisiones de las restricciones fuertes que no se pudieron resolver durante una ejecución de los diferentes algoritmos. Cada columna representa el algoritmo implementado para la resolución del problema y las filas representan el elemento al que se le asigna el horario.

Estos resultados muestran que las colisiones de Espacios son las más difíciles de resolver. Ésto es así, debido a que se contemplan 298 espacios para este problema, además esta cifra es mucho mayor que el número de profesores y grupos.

El algoritmo Hiper-Heurístico, a pesar de que su tiempo de ejecución mínimo promedio se acerca a la del algoritmo Genético (Figura 6.6), su resultado es mucho mejor que todos los algoritmos Meta-Heurísticos, sin contar que se pueden modificar los valores de los parámetros de entrada al utilizar el sistema y/o agregar otros algoritmos de búsqueda que no ocupen tanto tiempo como el algoritmo Genético (más de 6 horas). También es importante mencionar que los resultados mostrados en la Figura 6.5 son obtenidos a partir de los mejores valores de parámetros para cada tipo de



Elemento para el cual se está visualizando el horario ①

Espacios (685)

Total de Colisiones ②

– Aula (406)

* Aula 01 (8) -tabla +datos C 40 Proyector

HORARIO	Lunes	Martes	Miércoles	Jueves	Viernes
9:00 - 10:00	Cálculos Básicos en Ing. en Alimentos Mtr. Alma Yadira Salazar Govea 206-A (17) T		Cálculos Básicos en Ing. en Alimentos Mtr. Alma Yadira Salazar Govea 206-A (17) T	Cálculos Básicos en Ing. en Alimentos Mtr. Alma Yadira Salazar Govea 206-A (17) T	Cálculos Básicos en Ing. en Alimentos Mtr. Alma Yadira Salazar Govea 206-A (17) T
10:00 - 11:00	Introducción a la Auditoría Lic. Miguel Ángel Coronado Alcántara 605-B (26) T	Introducción a la Auditoría Lic. Miguel Ángel Coronado Alcántara 605-B (26) T Sistemas constructivos Lic. Jesús Sánchez Luqueño 603-A (36) T	Introducción a la Auditoría Lic. Miguel Ángel Coronado Alcántara 605-B (26) T Sistemas constructivos Lic. Jesús Sánchez Luqueño 603-A (36) T	Introducción a la Auditoría Lic. Miguel Ángel Coronado Alcántara 605-B (26) T Sistemas constructivos Lic. Jesús Sánchez Luqueño 603-A (36) T	Introducción a la Auditoría Lic. Miguel Ángel Coronado Alcántara 605-B (26) T
11:00 - 12:00		S mas constructivos Lic. Jes 603-A (36) T	Sistemas constructivos Sánchez Luqueñ 603-A (36) T		
12:00 - 13:00	Contabilidad I Lic. Martha A. Ruiz González 205-C (40) T	Contabilidad I Lic. Martha A. Ruiz Gon 205-C (40) T	Contabilidad I 205-C (40) T	Contabilidad I a A González 205-C (40) T	Contabilidad I Lic. Martha A. Ruiz González 205-C (40) T
16:00 - 17:00		Circuitos eléctricos I 414-A (29) T	Circuit ctricos I Mtr. Enrique Espinosa Justo 414-A (29) T	Circuitos eléctricos I 414-A (29) T	
18:00 - 19:00	Álgebra lineal Mtr. Vulfrano Tochihuitl Bueno 214-C (40) T Seminario de tesis II Dr. Julián Javier Carmona Rodríguez 1017-A (8) T	Álgebra lineal Mtr. Vulfrano Tochihuitl Bueno 214-C (40) T Seminario de tesis II Dr. Julián Javier Carmona Rodríguez 1017-A (8) T	Álgebra lineal Mtr. Vulfrano Tochihuitl Bueno 214-C (40) T Seminario de tesis II Dr. Julián Javier Carmona Rodríguez 1017-A (8) T	Álgebra lineal Mtr. Vulfrano Tochihuitl Bueno 214-C (40) T Seminario de tesis II Dr. Julián Javier Carmona Rodríguez 1017-A (8) T	Álgebra lineal Mtr. Vulfrano Tochihuitl Bueno 214-C (40) T Seminario de tesis II Dr. Julián Javier Carmona Rodríguez 1017-A (8) T

Asignaciones que tienen colisiones ③

Figura 6.4: Ejemplo de Colisiones en el Horario generado en el sistema.

BÚSQUEDA ARMÓNICA	ALGORITMO GENÉTICO	BÚSQUEDA TABÚ	RECOCIDO SIMULADO	HIPER-HEURÍSTICA
Grupos (527)	Grupos (482)	Grupos (409)	Grupos (390)	Grupos (452)
Profesores (313)	Profesores (321)	Profesores (270)	Profesores (284)	Profesores (264)
Espacios (728)	Espacios (570)	Espacios (685)	Espacios (632)	Espacios (279)

Figura 6.5: Colisiones para los Profesores con el algoritmo Genético.

algoritmo.

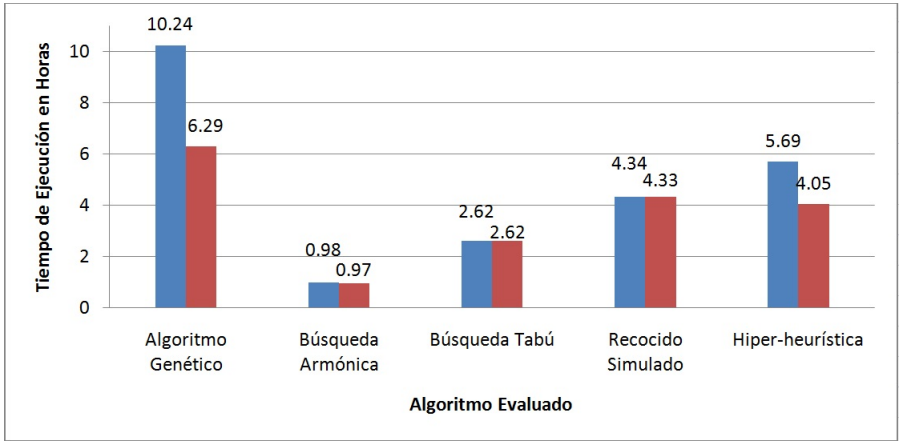


Figura 6.6: Resultado final del Tiempo de Ejecución en Horas para cada uno de los algoritmos.

En la Figura 6.6 se muestra la gráfica en donde se comparan los tiempos de ejecución de los algoritmos meta-heurísticos contra el algoritmo hiper-heurístico, en esta gráfica se muestran los tiempos promedio y mínimo de ejecución de cada algoritmo, contabilizando las 50 ejecuciones de cada combinación de parámetros del algoritmo para obtener los resultados mostrados, a excepción de la Hiper-Heurística, que solo tiene una combinación de parámetros (la mejor de Búsqueda Tabú). Además, se muestran los tiempos en horas, para que se pueda realizar una comparación mas visible.

El código realizado en Python está compartido a través de la plataforma de GitHub, el enlace se muestra en la sección de Apéndice.

Como resultado adicional se tiene un artículo en el libro de Software Libre que la Universidad realiza, en dicho artículo se menciona el modelo de restricciones del problema de Horarios de la UTM.

6.0.2. Discusiones

El algoritmo Hiper-Heurístico, por utilizar la Meta-Heurística de Recocido Simulado y de Búsqueda Armónica en sus búsquedas locales, tiende a tener un tiempo elevado con respecto a los algoritmos Hiper-Heurísticos que utilizan heurísticas como búsqueda local, sin embargo, se pueden utilizar menos iteraciones o romper los ciclos de estos algoritmos al tener cierto porcentaje de minimización. Este tiempo es elevado debido a el algoritmo Hiper-Heurístico resuelve a través del Recocido Simulado la restricción número 4 en cuánto se detecta que es el problema a resolver. Entonces, cada vez que el algoritmo principal resuelve el problema de la restricción 4, toma un poco menos del tiempo promedio del algoritmo de Recocido Simulado de la mejor combinación de parámetros para obtener una solución.

Tomando en cuenta el tiempo de ejecución promedio de los algoritmos, el algoritmo Genético es el mas tardado entre los cuatro algoritmos meta-heurísticos comparados en este trabajo de Tesis. Luego el algoritmo Hiper-Heurístico reduce casi a la mitad ese tiempo de ejecución y, en cuanto a función de evaluación, es mucho mejor que los demás algoritmos evaluados, se puede considerar como una buena opción para resolver el problema de Horarios de la UTM.

En la gráfica 6.6 también se observa que el tiempo promedio de ejecución del algoritmo de Búsqueda Armónica es aproximadamente de una hora, lo cual puede verse como un tiempo diez veces menor que la del algoritmo Genético. Sin embargo, el valor de función de factibilidad que se obtiene con este algoritmo no es lo suficientemente bueno como para utilizarla en la resolución y generación del horario de clases de la UTM.

Hay que tener en cuenta que no se probaron las ejecuciones de los algoritmos con valores mayores a 500 ejecuciones por falta de tiempo en las pruebas. El tiempo de las pruebas depende de las combinaciones de parámetros y en mayor medida en las ejecuciones o tamaño de población.

Para que haya un poco más de consistencia en el trabajo, es necesario resolver los problemas externos o técnicos que pudieran surgir al realizar algún otro trabajo de Calendarización o en su caso alguna modificación. Éstos problemas a resolver son principalmente la falta de información en el momento requerido, por ejemplo: No se capturan todos los datos de manera correcta; Hay profesores que están ingresando o en proceso de ingresar y

se necesita que se les asigne materias, etc.

Los algoritmos utilizados en este trabajo son bastante buenos resolviendo diferentes problemas de optimización, sin embargo, hace falta adaptar o buscar un algoritmo que se adecúe de mejor manera al problema de Horarios de la UTM. Lo que puede realizarse en un tiempo futuro.

Para que el algoritmo obtenga mejores resultados, es necesario preprocesar los datos para restringir detalles que puedan causar algún conflicto al buscar una solución, en este enlace[1] se puede apreciar un sistema de Calendarización para Universidades que utiliza pre-procesamiento y en la página se argumenta que obtienen resultados factibles.

En [20] resuelven un problema real de calendarización de una Universidad utilizando un algoritmo de enfoque constructivo, en dicho trabajo se menciona que el tiempo mínimo requerido para encontrar una solución factible es desde 3 horas con 33 minutos hasta 6 horas con 22 minutos, así, se confirma que los problemas reales necesitan gran tiempo de procesamiento.

En el siguiente enlace[2] se tiene una descripción de cómo se puede resolver el problema de calendarización utilizando un algoritmo evolutivo cuántico. Es solo cuestión de tiempo que se tenga una computadora cuántica y el problema de calendarización podría resolverse en poco tiempo y con resultados óptimos en teoría.

[1] http://www.mathplan.de/en/

[2] http://www.open-science-repository.com/application-of-quantum-evolutionary-algorithm-to-complex-timetabling-problem.html

Capítulo 7

Conclusiones y Trabajo Futuro

En este trabajo de Tesis se presenta una propuesta de una Hiper-Heurística basada en el algoritmo de Búsqueda Tabú para resolver el problema de calendarización de horarios de la Universidad Tecnológica de la Mixteca. Esta propuesta se realiza a partir de la comparación de cuatro algoritmos Meta-Heurísticos, las cuales son: Algoritmo Genético, Algoritmo de Recocido Simulado, Algoritmo de Búsqueda Tabú y Algoritmo de Búsqueda Armónica.

El algoritmo Hiper-Heurístico obtuvo un buen desempeño en cuanto al valor de función de evaluación comparado con los algoritmos Meta-Heurísticos presentados. Sin embargo, el tiempo de ejecución de la Hiper-heurística no fue lo más bueno debido a la manera en cómo se implementó. La elección de un algoritmo depende tanto del tiempo de ejecución, así como del valor de función de evaluación que dé como resultado. No se puede aceptar un algoritmo que rebase el tiempo en que se realiza el horario de manera manual y menos si éste da un valor de función de evaluación relativamente alta. Aunque la Hiper-Heurística obtiene resultados en un tiempo promedio de 5.6 horas; tiempo que es mucho menor al que ocupan al realizar el horario de manera manual; su valor de función de evaluación sigue siendo bastante alta a la deseada.

La hiper-heurística puede utilizarse de manera parcial en la creación de un Horario de Clases para la UTM para recomendar a los responsables del mismo una buena solución desde la cual iniciar con el proceso. En un futuro, realizando algunas experimentaciones y modificaciones en los parámetros, se pueden obtener nuevas soluciones que mejoren el resultado que se obtiene actualmente en este trabajo.

Viendo el problema desde el punto de vista del encargado que realiza las asignaciones de horarios en la UTM, utilizar el sistema con el algoritmo hiper-heurístico puede ayudar a disminuir mucho trabajo que se debería realizar de manera manual, ya que se reduce un 50 % de las restricciones aproximadamente, lo cual es un buen avance en el trabajo.

Además, si se realizan modificaciones en los valores de los parámetros de entrada, o se agregan algoritmos de búsqueda diferentes en la búsqueda de la hiper-heurística como por ejemplo, los algoritmos de inserción o los algoritmos híbridos entre heurísticas y meta-heurísticas, se podrían obtener mejores resultados y reducir aún más la función de evaluación.

Es importante tomar en cuenta que no se puede reducir la función de evaluación al 100 % debido a los diferentes problemas que surgen en la práctica. Por ejemplo, en la base de datos existen datos incompletos o hay asignaciones que siempre van a tener colisiones porque pueden tener los mismos espacios de búsqueda.

Como las pruebas realizadas en este trabajo de Tesis son para fines comparativos, se pueden realizar los siguientes cambios en un futuro, de manera que se pueda utilizar el trabajo para obtener mejores soluciones o solucionar completamente el problema:

- Se puede intentar solucionar el problema con otras meta-heurísticas o variantes de las ya descritas.
- Se pueden modificar los parámetros de las meta-heurísticas y la hiper-heurística para encontrar mejores soluciones.
- Implementar una Hiper-Heurística basada en el Algoritmo Genético, siempre teniendo en cuenta el tiempo.

- Dividir el problema en tres partes, primero se pueden descartar las asignaciones fijas (que contienen los datos estáticos) desde el inicio, luego encontrar una población de soluciones diferentes que resuelvan las asignaciones prioritarias (contienen datos prioritarios). Y, basados en esas soluciones, implementar alguno de los algoritmos descritos previamente para lidiar con las asignaciones restantes, las cuales generalmente son las que generan los problemas de asignación.

- Combinar con redes neuronales el espacio de heurísticas que utiliza la hiper-heurística. Se puede entrenar una red neuronal basándose en los horarios que se han tenido o en caso de ser necesario, a partir de unos buenos resultados del algoritmo hiper-heurístico presentado en este trabajo. Así la red neuronal generaría soluciones similares o mejores a las presentadas en este trabajo.

- Para mejorar el tiempo de ejecución, se pueden buscar otras maneras de modelar la solución, de tal manera que se puedan disminuir los datos a evaluar. Se puede inicializar el algoritmo con *datos preprocesados*, este proceso se agregaría después de la extracción de datos y puede consistir en obtener los datos fijos, además de resolver las colisiones que puedan haber en los datos fijos, por ejemplo: si dos profesores deben dar clases en una aula, a una hora y día específicos, entonces el creador del horario debe resolver con ambos profesores la colisión manualmente.

- Codificar en C++ en lugar de Python para obtener más eficiencia en cuánto al tiempo de ejecución, así se pueden realizar experimentaciones con valores de parámetros elevados.

- Se puede intentar resolver el problema dividiendo los tres tipos de elementos que se les asigna el horario (Profesor, Grupo y Espacio), de tal manera que por cada elemento se debe encontrar una buena solución, tomando en cuenta que los tres elementos son dependientes.

- Se puede realizar un reajuste del tamaño del problema, de tal manera que se pueda ir solucionando el problema empezando por un tamaño pequeño de datos de entrada, para luego ir aumentando el

tamaño y de ésta manera ir mejorando o buscando nuevos parámetros o algoritmos para la resolución del problema total.

Apéndice A

Código Fuente

El código fuente se encuentra alojado en el siguiente enlace: `https://github.com/kamjotpetjaay/HiperheuristicaTabu`. No se anexa de manera física debido a que es extenso para este documento.

Bibliografía

[1] Esraa Abdelhalim and Ghada El Khayat. A Utilization-based Genetic Algorithm for Solving the University Timetabling Problem (UGA). 55:1395–1409, 03 2016.

[2] Salwani Abdullah, Edmund K. Burke, and Barry Mccollum. An investigation of variable neighbourhood search for university course timetabling. pages 413–427, 2005.

[3] Salwani Abdullah, Khalid Shaker, Barry Mccollum, and Paul Mcmullan. Construction of course timetables based on great deluge and tabu search. 01 2010.

[4] Mohammed Azmi Al-betar Al-balqa. A harmony search algorithm for university course timetabling. (April), 2012.

[5] Mohammed Al-Betar and Ahamad Tajudin Khader. A harmony search algorithm for university course timetabling. 194:3–31, 04 2012.

[6] Mohammed Azmi Al-Betar and Ahamad Tajudin Khader. A harmony search algorithm for university course timetabling. *Annals of Operations Research*, 194(1):3–31, 2012.

[7] Cagdas Aladag, Gulsum Hocaoglu, and Murat Basaran. The effect of neighborhood structures on tabu search algorithm in solving course timetabling problem. 36:12349–12356, 12 2009.

[8] Asaju La aro Bolaji, Ahamad Tajudin Khader, Mohammed Azmi Al-Betar, and Mohammed A. Awadallah. University course timetabling

using hybridized artificial bee colony with hill climbing optimizer. *Journal of Computational Science*, 5(5):809–818, 2014.

[9] Masri Ayob, Ariff Malik, Salwani Abdullah, Abdul Hamdan, Graham Kendall, and Rong Qu. Solving a Practical Examination Timetabling Problem: A Case Study. *Computational Science and Its Applications – ICCSA 2007*, 4707:611–624, 2007.

[10] Hamed Babaei, Jaber Karimpour, and Amin Hadidi. A survey of approaches for university course timetabling problem. *Computers and Industrial Engineering*, 86(April):43–59, 2014.

[11] Ruggero Bellio, Sara Ceschia, Luca Di, Andrea Schaerf, and Tommaso Urli. Feature-based tuning of simulated annealing applied to the curriculum-based course timetabling problem. *Computers and Operation Research*, 65:83–92, 2016.

[12] Peter Brucker, Johann Hurink, and Frank Werner. Improving local search heuristics for some scheduling problems. Part II. *Discrete Applied Mathematics*, 72(1-2):47–69, 1997.

[13] Edmund Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

[14] Edmund K. Burke, Barry McCollum, Amnon Meisels, Sanja Petrovic, and Rong Qu. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 176(1):177–192, 2007.

[15] Michael W. Carter and Gilbert Laporte. The practice and theory of automated timetabling: Selected papers from the 1st international conference. *Lecture Notes in Computer Science*, 1153:3–21, 1996.

[16] Josu Ceberio, Alexander Mendiburu, and Jose A. Lozano. The linear ordering problem revisited. *European Journal of Operational Research*, 241(3):686–696, 2015.

[17] Konstantin Chakhlevitch and Peter Cowling. Hyperheuristics: Recent developments. *Adaptive and Multilevel Metaheuristics*, 136:3–29, 2008.

[18] P. C. Chen, G. Kendall, and G. V. Berghe. An ant based hyper-heuristic for the travelling tournament problem. In *2007 IEEE Symposium on Computational Intelligence in Scheduling*, pages 19–26, April 2007.

[19] Tim B. Cooper and Jeffrey H. Kingston. The complexity of timetable construction problems. In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling*, pages 281–295, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[20] Marco Cruz Chávez, Mireya Flores-Pichardo, Alina Martinez, Pedro Moreno-Bernal, and Martín H. Cruz-Rosales. Solving a real constraint satisfaction model for the university course timetabling problem: A case study. 2016:1–14, 01 2016.

[21] D. de Werra. *Some combinatorial models for course scheduling.* Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[22] S. Even, Alon Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4):691–703, 1976.

[23] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1990.

[24] Pablo Garrido and María Cristina Riff. Collaboration between hyper-heuristics to solve strip-packing problems. In Patricia Melin, Oscar Castillo, Luis T. Aguilar, Janusz Kacprzyk, and Witold Pedrycz, editors, *Foundations of Fuzzy Logic and Soft Computing*, pages 698–707, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[25] Pablo Garrido and María-Cristina Riff. An evolutionary hyperheuristic to solve strip-packing problems. In Hujun Yin, Peter Tino, Emilio Corchado, Will Byrne, and Xin Yao, editors, *Intelligent Data Engineering and Automated Learning - IDEAL 2007*, pages 406–415, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[26] Hahn goldberg Shoshana. Defining, modeling, and solving a real university course timetabling problem. Master's thesis, University of Toronto, 2007.

[27] Emma Hart and Peter Ross. A heuristic combination method for solving job-shop scheduling problems. In Agoston E. Eiben, Thomas Bäck, Marc Schoenauer, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature — PPSN V*, pages 845–854, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[28] Frederick S Hillier. *Handbook of Metaheuristics*, volume 157. 2010.

[29] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.

[30] S. F. H. Irene, S. Deris, and M. H. S. Zaiton. A study on pso-based university course timetabling problem. In *2009 International Conference on Advanced Computer Control*, pages 648–651, Jan 2009.

[31] Graham Kendall and Naimah Mohd Hussin. An investigation of a tabu-search-based hyper-heuristic for examination timetabling. In Graham Kendall, Edmund K. Burke, Sanja Petrovic, and Michel Gendreau, editors, *Multidisciplinary Scheduling: Theory and Applications*, pages 309–328, Boston, MA, 2005. Springer US.

[32] Berna Kiraz, A. Şima Etaner-Uyar, and Ender Özcan. An ant-based selection hyper-heuristic for dynamic environments. In Anna I. Esparcia-Alcázar, editor, *Applications of Evolutionary Computation*, pages 626–635, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[33] Philipp Kostuch. The university course timetabling problem with a three-phase approach. pages 109–125, 01 1970.

[34] G.K. Koulinas and K.P. Anagnostopoulos. A new tabu search-based hyper-heuristic algorithm for solving construction leveling problems with limited resource availabilities. *Automation in Construction*, 31:169 – 175, 2013.

[35] Pacheco Agüero Carla Leninca. Distribución Óptima de Horarios de Clases utilizando la técnica de Algoritmos Genéticos. Master's thesis, Universidad Tecnológica de la Mixteca, 2000.

[36] April Lin Lovelace. On the Complexity of Scheduling University Courses. Master's thesis, Faculty of California Polytechnic State University, 2010.

[37] S. Mary, Saira Bhanu, and N. P. Gopalan. A Hyper-Heuristic Approach for Efficient Resource Scheduling in Grid. III(3):249–258, 2008.

[38] Barry Mccollum, Andrea Schaerf, Ben. Paechter, Paul Mcmullan, Rhydian Lewis, Andrew J Parkes, Luca Di Gaspero, Rong Qu, and Edmund Burke. Setting the Research Agenda in Automated Timetabling : The Second International Time- tabling Competition. *INFORMS Journal on Computing*, 22(May):120–130, 2010.

[39] Abraham Duarte Muñoz. *Metaheurísticas*. Ciencias Experimentales y Tecnolog¡a. Editorial Dykinson, S.L., 2007.

[40] Nelishia Pillay. A survey of school timetabling research. *Annals of Operations Research*, 218(1):261–293, 2014.

[41] Michael Lawrence Pinedo. *Scheduling: Theory, Algorithms, and Systems, fifth edition*. 2016.

[42] Lewis Rhyd and J. Thompson. Analysing the effects of solution space connectivity with an effective metaheuristic for the course timetabling problem. *European Journal of Operational Research*, 240(3):637–648, 2014.

[43] Tomás Robenek, Jianghang Chen, and Bierlaire Michel. The Ideal Train Timetabling Problem The Ideal Train Timetabling Problem. *20th Conference of the International Federation of Operational Research Societies*, (May):1–15, 2014.

[44] A. Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13(2):87–127, 1999.

[45] Modesto Seara Vásquez. *Un nuevo Modelo de Universidad*. Number 2. 2010.

[46] Krzysztof Socha, Michael Sampels, and Max Manfrin. Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. 334-345, 02 2003.

[47] Jorge A. Soria-Alcaraz, Ender Özcan, Jerry Swan, Graham Kendall, and Martin Carpio. Iterated local search using an add and delete hyper-heuristic for university course timetabling. *Applied Soft Computing Journal*, 40:581–593, 2016.

[48] Chong Keat Teoh, Antoni Wibowo, and Mohd Salihin Ngadiman. Review of state of the art for metaheuristic techniques in Academic Scheduling Problems. *Artificial Intelligence Review*, 44(1):1–21, 2015.

[49] Thatchai Thepphakorn, Pupong Pongcharoen, and Chris Hicks. Modifying regeneration mutation and hybridising clonal selection for evolutionary algorithms based timetabling tool. *Mathematical Problems in Engineering*, 2015, 2015.

[50] Juliana Wahid and Juliana Wahid. Harmony Search Algorithm for Curriculum- Based Course Timetabling Problem Harmony Search Algorithm for Curriculum-Based Course Timetabling Problem. *CoRR*, abs/1401.5156(January 2015), 2014.

[51] Anthony Wehrer and Jay Yellen. The design and implementation of an interactive course-timetabling system. *Annals of Operations Research*, 218(1):327–345, 2014.

[52] J. Xie, Y. Mei, A. T. Ernst, X. Li, and A. Song. A genetic programming-based hyper-heuristic approach for storage location assignment problem. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 3000–3007, July 2014.

[53] Shengxiang Yang and Sadaf Naseem Jat. Genetic algorithms with guided and local search strategies for university course timetabling. 41:93 – 106, 01 2011.

[54] Stelios H Zanakis and James R Evans. Heuristic .ºptimization": Why , When , and How to Use It WHY , WHEN , AND HOW TO USE IT. *Interfaces*, 11(5)(August 2015):84–91, 1981.