Casa abierta al tiempo

UNIVERSIDAD AUTÓNOMA METROPOLITANA

Unidad Iztapalapa

DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA

“ASIGNACIÓN DE SALONES POR MEDIO DE UNA HIPER-HEURÍSTICA”

TESIS

QUE PRESENTA

JORGE CARLOS GONZÁLEZ GONZÁLEZ

MATRÍCULA 2183802139

PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS Y TECNOLOGÍAS DE LA INFORMACIÓN

DIRECTORA: M.C. ALMA EDITH MARTÍNEZ LICONA

JURADO:

PRESIDENTE: DR. JORGE ALBERTO SORIA ALCARAZ

SECRETARIO: DR. ERIC ALFREDO RINCÓN GARCÍA

VOCAL: M.C. ALMA EDITH MARTÍNEZ LICONA

Iztapalapa, Ciudad De México junio 2021

Resumen

El problema de horarios y cursos basado en currículum (abreviado por sus siglas en inglés, CBCT), es un problema de optimización, donde se plantea la generación de un calendario escolar respetando una serie de restricciones, además existe una función objetivo con la capacidad de evaluar cada horario propuesto, por lo que el objetivo es obtener el calendario con el menor costo posible.

Los origines del problema pueden ser rastreados hasta los años setentas, aunque en el presente trabajo se considera la descripción dada por la Competencia Internacional de Horarios 2007 (por sus siglas en inglés: ITC2007), evento donde se reunieron investigadores alrededor del mundo y que continúa siendo utilizado como campo de estudio para algoritmos.

En el presente trabajo se propone una hiper-heurística como técnica para abordar el CBCT. El algoritmo por medio de diferentes heurísticas de bajo nivel, intenta minimizar el número de restricciones no satisfechas con el objetivo de generar un calendario de mejor calidad.

Finalmente se utilizó la base de datos de la ITC2007 la cual consta de 21 instancias distintas con lo cual, se puede tener marco de referencia sobre el desempeño de la propuesta. Los resultados obtenidos por el algoritmo, son comparados con otras técnicas encontradas en la literatura. Los resultados obtenidos son alentadores, el programa obtiene soluciones competitivas en tiempos aceptables, e incluso en algunos casos cercanas al mejor valor conocido.

Agradecimientos

A la Universidad Autónoma Metropolitana unidad Iztapalapa por proporcionarme la oportunidad de realizar un posgrado en su magna casa de estudios.

Al Consejo Nacional de Ciencia y Tecnología por apoyarme ha realizar mis estudios de posgrado.

Un gran agradecimiento a mi asesora la M.C. Alma Edith Martínez Licona por su apoyo brindado durante toda la realización de mis estudios de posgrado.

A Lucero Medina Espinoza, por todo su amor, cariño y paciencia que ha tenido conmigo durante cada etapa del posgrado.

A mi madre Juana y hermana Melissa, por ser mis pilares de ayuda y consejo.

Para todos aquellos con quienes he podido compartir esta experiencia de vida. Para todos aquellos que me han expresado su apoyo, amistad y cariño.

Índice

Resumen ..... **i**

Agradecimientos ..... **ii**

Índice ..... **iii**

Abreviaturas ..... **v**

Ilustraciones ..... **vi**

Tablas ..... **vi**

Metodología ..... **vii**

Estructura de la ICR ..... **ix**

1. Introducción **1**
1.1 Justificación 4
1.2 Objetivo general 5
1.3 Objetivo especifico 6
1.4 Preguntas de investigación 6
1.5 Hipótesis 6

2. Inteligencia Artificial **7**
2.1 Optimización 9
2.2 Heurística 11
2.3 Metaheurísticas 12
2.3.1 Explotación y exploración 15
2.3.2 Enfoque de solución única y basado en población 16
2.3.3 Función objetivo 18
2.3.4 Condición de paro 19

3. Hiper-heurísticas **21**
3.1 Nivel alto y nivel bajo 23
3.2 Clasificación de enfoque hiper-heurísticos 25
3.2.1 Hiper-heurística selección-constructiva 25
3.2.2 Hiper-heurística de selección perturbación 26
3.2.3 Hiper-heurística de generación-construcción 28

3.2.4 Hiper-heurística generación-perturbación 29
3.3 Aprendizaje en las hiper-heurísticas 30
3.4 El desafío heurístico de búsqueda de dominio cruzado y HyFlex 32
3.5 Aplicaciones 35
3.5.1 El problema de enrutamiento de vehículos 35
3.5.2 Problemas de horarios para personal 36
3.5.3 Problemas de horarios para instituciones educativas 36

4. El problema de horarios y cursos basado en currículum 38
4.1 Historia 41
4.2 Descripción de el problema de horarios y cursos basados en currículum 42
4.2.1 Variables 43
4.2.2 Restricciones 44
4.2.3 Evaluación 47
4.3 Propuestas para abordar el problema de horarios y cursos basados en currículum 48
4.3.1 Recocido simulado 48
4.3.2 Gran diluvio 51
4.3.3 Algoritmos genéticos 52
4.3.4 Hiper-heurísticas 55
4.3.5 Algoritmos híbridos 56

5. Algoritmo propuesto 58
5.1 Representación de la solución 58
5.2 Generación de una solución inicial 60
5.3 Función de evaluación 62
5.4 Nivel alto de la hiper-heurística 62
5.5 Mecanismo de selección para las heurísticas de bajo nivel 63
5.6 Heurísticas de bajo nivel 66
5.7 Condición de paro 68

6. Experimentación 69
6.1 Instancias 69
6.2 Parámetros del algoritmo propuesto 71
6.3 Consideraciones adicionales 72

7. Resultados 74

8. Conclusiones 84
8.1 Trabajo a futuro 86

Anexo: Pseudocódigo 98

Abreviaturas

Abreviatura	Término
CBCT	El problema de horarios y cursos basado en currículum
CHeSC	El desafío heurístico de búsqueda de dominio cruzado
ETP	El problema de horarios para exámenes
HHGC	Hiper-heurística generación-construcción
HHGP	Hiper-heurística generación-perturbación
HHSC	Hiper-heurística selección-construcción
HHSP	Hiper-heurística selección-perturbación
HyFlex	Librería flexible para hiper-heurísticas
IA	Inteligencia artificial
ITC	Competencia Internacional de Horarios
UCTP	El problema de horarios y cursos universitarios
VRP	El problema de enrutamiento de vehículos

Ilustraciones

Ilustración 1. Representación de la solución. 59
Ilustración 2. Inserción de cursos. 59
Ilustración 3. Resultados de las instancias comp01 a la comp10. 76
Ilustración 4. Resultados de las instancias comp11 a la comp21. 76
Ilustración 5. Resultados de la instancia comp01 a la comp05. 80
Ilustración 6. Resultados de la instancia comp06 a la comp10. 80
Ilustración 7. Resultados de la instancia comp11 a la comp15. 81
Ilustración 8. Resultados de la instancia comp16 a la comp21. 81
Ilustración 9. Desviación estándar comp01 a comp10. 82
Ilustración 10. Desviación estándar comp11 a comp21. 82

Tablas

Tabla 1. Descripción de la base de datos ITC2007. 70
Tabla 2. Tabla de valores obtenidos por las técnicas. 78

Metodología

El tema de trabajo que se desarrolló en el presente documento consiste en el diseño e implementación de una hiper-heurística aplicada al problema de horarios y cursos basados en currículum. Además, se realizaron una serie de experimentos para evaluar desempeño del algoritmo, adicionalmente se efectuó una comparación contra otras propuestas encontradas en la literatura y finaliza el texto con las conclusiones obtenidas. Por lo que, para lograr la investigación, se realizó la siguiente metodología, la cual consta de cinco etapas importantes.

1) Revisión del estado del arte: en esta fase se realizó una búsqueda en la literatura especializada, con el fin de encontrar las últimas propuestas realizadas para abordar el problema CBCT. Particularmente se concentraron los esfuerzos en algoritmos metaheurísticos e hiper-heurísticas, con el fin de analizar más detenidamente dichas estrategias.

2) Diseño e implementación del algoritmo: durante la etapa de revisión del estado del arte, se analizaron detenidamente algunos artículos (particularmente los que utilizaron hiper-heurísticas) con el fin de identificar mecanismo y métodos, para encontrar posibles áreas de oportunidad para la investigación.

Con la información obtenida, se procedió a la fase de diseño e implementación del algoritmo, donde se programó la estructura de la hiper-heurística, así como también las heurísticas de bajo nivel que se emplearon para abordar CBCT.

3) Experimentación: se emplearon las instancias proporcionadas por la Competencia Internación de Horarios 2007, la cual es un referente para el problema CBCT. Se realizaron varias pruebas con la base de datos con el objetivo de obtener

los valores más adecuados para cada parámetro del algoritmo para abordar la tarea.

Con los valores determinados para los parámetros del algoritmo, se procedió a la etapa de experimentación. Adicionalmente se repitieron las pruebas en cada instancia, con el fin de obtener datos estadísticos del comportamiento de la hiper-heurística, el desempeño para el problema CBCT y el rendimiento particular en cada instancia.

4) Análisis y resultados: Obtenidos los datos, se procedió a realizar una comparación con otras estrategias metaheurísticas e hiper-heurísticas encontradas en la literatura, con la finalidad de equiparar el desempeño.

5) Conclusiones: Al final, la investigación cierra con las conclusiones y la escritura de la ICR.

Estructura de la ICR

El presente documento se encuentra organizado de la siguiente manera: en el Capítulo 1, se comienza con la introducción del documento, la justificación del proyecto de investigación, los objetivos y las preguntas de investigación.

En el Capítulo 2, se habla acerca de la inteligencia artificial, así como también las aplicaciones que ha tenido para abordar problemas de diferentes índoles. Después se aborda con detenimiento las principales características de los problemas de optimización, así como algunas de las estrategias empleadas para resolverlos.

Posteriormente en el Capítulo 3, se explica con detalle los algoritmos hiper-heurísticos, sus características propias, la estructura interna que poseen, su clasificación y las aplicaciones a problemas.

Después en el Capítulo 4 se describe el problema de horarios y cursos basados en currículum, comenzando por el origen del problema en la Competencia de Internacional de Horarios del año 2007. Luego se explica con mayor detenimiento sobre sus características: variables, restricciones duras, restricciones blandas y evaluación de solución. Para cerrar el capítulo se mencionan algunos algoritmos empleados para resolver la tarea.

Luego en el Capítulo 5 se presenta el algoritmo empleado para abordar CBCT, explicando de manera detallada el funcionamiento, mecanismos empleados, la estructura de la hiper-heurística y la generación de la solución inicial de la que parte la propuesta.

En el Capítulo 6 se hablan de la experimentación realizada; parámetros de la técnica, cantidad de pruebas realizadas, tiempo de ejecución permitido para el algoritmo, así como otra serie de consideraciones para la experimentación.

Después, en el Capítulo 7 se muestran los resultados obtenidos por la estrategia propuesta, así como una serie de comparaciones con otros algoritmos encontrados en la literatura que también intentan resolver la tarea.

Para terminar, en el Capítulo 8 se encuentran las conclusiones obtenidas de la investigación, así como también el trabajo a futuro que se puede realizar. Después hay un apartado con la bibliografía utilizada en el documento y un anexo con el pseudocódigo del programa.

Capítulo 1

1. Introducción

Los problemas de optimización combinatoria son excelentes campos de estudios y pruebas para propuestas de algoritmos y heurísticas [1]; proveen de restricciones que deben ser satisfechas, las respuestas no suelen ser triviales, por último, las posibles soluciones factibles son bastas, pero sólo se desea la que proporcione los mejores beneficios. Cuando las ciencias computacionales se enfrentan a una tarea que, dadas sus características, no permite su resolución por algoritmos conocidos, ya sea porque no es viable en términos de tiempo o recursos de cómputo, se abre una ventana al desarrollo de nuevas técnicas y estrategias computaciones [2].

Desde los comienzos de la computación, el concepto de generar algoritmos que puedan ser calificados como inteligentes, ha sido una tarea a la que se le ha dedicado una gran cantidad de estudios y análisis, con el propósito de elaborar un programa que tenga la capacidad de razonamiento necesaria para solucionar problemas [3] [4].

Particularmente para las ciencias de la computación, los comportamientos inteligentes en algoritmos pueden ser representados como: análisis de datos, adaptación a cambios, resolución de tareas, flexibilidad para trabajar con información imprecisa o incompleta, aunque no solo se limita a los ejemplos descritos. La rama de las ciencias de la computación encargada de proponer esquemas con las propiedades descritas anteriormente es la inteligencia artificial [3][4].

La expresión *inteligencia artificial* fue sugerida por J. McCarthy, profesor de la Universidad de Stanford, en 1955 en una propuesta de investigación en el documento *Dartmouth Summer Research Project on Artificial Intelligence* junto con Marvin Minsky, Nat Rochester y Claude Shannon, para referirse al diseño de máquinas con la capacidad de realizar tareas humanas habitualmente catalogadas como inteligentes [4].

Como se ha mencionado la inteligencia artificial, es el campo de las ciencias computaciones encargado de estudiar y proponer técnicas de cómputo con la capacidad de realizar actividades propias de los seres humanos clasificadas como inteligentes. Gracias al perseverante desarrollo de la inteligencia artificial, se ha logrado generar una gran gama de propuestas con diversos enfoques que intentan abordar tareas complejas [1] [2].

El constante desarrollo e investigación de la inteligencia artificial ha generado diversas estrategias para abordar problemas, donde cada técnica posee sus características propias, así como una serie de pautas para su implementación y uso [1] [2]. Con el tiempo, algunos de los planteamientos se han convertido en áreas de estudio particulares de la inteligencia artificial, por mencionar algunos ejemplos se tiene: sistemas de partículas [5], algoritmos evolutivos [6], agentes inteligentes [7], redes neuronales [5], sistemas difusos [8], entre otros [1].

Dentro de los algoritmos sugeridos en la inteligencia artificial, existe una propuesta que ha llamado la atención por su flexibilidad para resolver distintos problemas de búsqueda y optimización con resultados competitivos. Las hiper-heurísticas [9] [10] son una estrategia que intenta obtener un nuevo nivel de generalidad al que poseen otros esquemas metaheurísticos. Realizando búsquedas sobre un espacio heurístico en lugar del espacio solución habitual [11] [12].

Si se analiza la palabra “hiper-heurística” se encuentra el prefijo “hiper” el cual denota superioridad, además de la palabra “heurística”, la cual es una técnica empleada en las ciencias de la computación para abordar problemas en donde se pretende generar respuestas aceptables sin comprometer los recursos computacionales [1][2].

Los orígenes de las hiper-heurísticas pueden ser rastreados hasta los años 60s [12], siendo hasta los años 2000 donde la técnica fue denominada como una: “*heurística que selecciona otra heurística*” [12]. Una de las características más sobresalientes de las hiper-heurísticas es la capacidad de explorar sobre el espacio heurístico al clásico espacio solución [9]. Lo anterior es posible gracias a que tienen a su disposición un conjunto de heurísticas de bajo nivel, las cuales trabajan directamente con el problema, por lo que el algoritmo busca cuál estrategia es la idónea para abordar el problema [10] [11].

No existe una metodología para elegir el tipo de heurísticas de bajo nivel ni el número de las mismas, por lo que el investigador es quién decide esas propiedades. Lo anterior con base al problema de estudio y los intereses particulares de la investigación [13].

Cada heurística de bajo nivel es evaluada durante el proceso de optimización, y dependiendo de los resultados obtenidos en el desarrollo de la búsqueda, tendrán mayor posibilidad de ser seleccionada en el futuro. Las heurísticas de bajo nivel que no han tenido resultados adecuados, también tienen la oportunidad de participar en otras etapas del problema, donde tal vez se puedan obtener un mejor desempeño (particularmente el ejemplo descrito es una hiper-heurística con retroalimentación en línea) [14].

Son las características de adaptabilidad y modularidad las que hacen a las hiper-heurísticas que sean ampliamente estudiadas, documentadas y aplicadas a una serie de diversos problemas de optimización como lo pueden ser: permutación

para flujo de tienda (*permutation flow shop*) [9], enrutamiento para grupo de vehículos (*vehicle routing grouped*) [15] o la generación de horarios para instituciones educativas. Particularmente para estos últimos, han tenido una buena recepción como método para resolver la tarea [8].

Para las instituciones educativas, la formación de horarios es un problema que deben enfrentar en cada ciclo escolar. En cada nuevo periodo escolar se deben de reajustar o generar calendarios en los que se contemplen una diversa cantidad de elementos como lo son: profesores, alumnos, cursos, aulas, eventos escolares, etc. Por lo que generar un calendario donde se pueda integrar todos los componentes no es una tarea trivial [8].

1.1 Justificación

Los problemas de horarios surgen en diversos campos donde se tenga la necesidad de administrar diversos recursos (infraestructura, personas, tiempo, etc.) e.g. hospitales (*nurse rostering problem*) [9], compañías (*personnel scheduling*) [15], instituciones educativas, por mencionar algunos.

Particularmente para las instituciones universitarias es un problema de interés, debido a que cada nuevo periodo escolar, deben de generar o reajustar sus horarios para ofrecer una serie de asignaturas y eventos en los salones que poseen durante el periodo escolar. Es en este tipo de escenario donde entra el problema de horarios y cursos basados en currículum [16].

El problema de horarios y cursos basados en currículum (por sus siglas en inglés, CBCT) tiene como objetivo la administración de varios conjuntos de elementos (profesores, alumnos, cursos y clases) en periodos de tiempo para una serie de salones. Los recursos son finitos y están limitados por una serie de restricciones blandas y duras impuestas por el problema [17].

El objetivo de CBCT es generar un horario válido, asignando todos los cursos en los lapsos de tiempo disponible y en una serie de habitaciones viables, satisfaciendo el mayor número posible de restricciones, con la finalidad de aprovechar de la manera más adecuada los recursos de una institución educativa [17].

Las restricciones duras son de carácter obligatorio y todas deben ser cumplidas al momento de presentar un calendario. Mientras que las restricciones suaves son de carácter opcional, por lo que no es necesario cumplir cada una de ellas. Sin embargo, se busca satisfacer el mayor número posible de restricciones blandas, debido a que tiene una penalización en la aptitud del horario propuesto. Por lo que un horario es factible sí todas las restricciones duras son cumplidas, además será de mejor calidad cuantas más restricciones suaves satisfaga [17] [18].

Finalmente, la calidad de un calendario es un valor determinado por una función objetivo. En la estructura de la función objetivo, se ponderan todas las restricciones duras y blandas no satisfechas en el calendario [17].

Se ha seleccionado la técnica de las hiper-heurísticas, por sus propiedades de flexibilidad y adaptabilidad a los problemas [9]. Destacando la característica de poseer un conjunto de diversas heurísticas de bajo nivel. Además de que ha sido empleada en otros problemas de índole similar teniendo resultados competitivos [8].

1.2 Objetivo general

Proponer un algoritmo hiper-heurístico capaz de asignar de forma adecuada salones a una serie de cursos bajo un conjunto de restricciones.

1.3 Objetivo especifico

Analizar los algoritmos encontrados en la literatura para abordar el problema de CBCT.

Proponer un algoritmo hiper-heurístico para abordar el problema CBCT.

Comparar los resultados del algoritmo propuesto con otros esquemas de índole similar encontrados en la literatura.

1.4 Preguntas de investigación

¿Qué estrategias se emplean para resolver el problema de CBCT?

¿Cuáles son las ventajas al emplear un esquema hiper-heurístico para abordar el problema CBCT?

¿Cuál es el desempeño del algoritmo propuesto comparado con otras estrategias encontradas en la literatura?

1.5 Hipótesis

El uso de una hiper-heurística para el problema de horarios y cursos basados en currículum, permite tener un alto grado de adaptabilidad para cada instancia presentada. Lo que permite tener resultados competitivos para el problema de estudio, comparado contra otras estrategias en la literatura.

Capítulo 2

2. Inteligencia Artificial

En la actualidad, la inteligencia artificial (IA) ha tomado un mayor impulso en su desarrollo e implementación como estrategia para abordar problemas de toda índole. Al día de hoy se pueden encontrar aplicaciones en áreas de ingeniería, la industria, finanzas e inclusive en herramientas dentro de nuestros teléfonos celulares para facilitar las tareas cotidianas [3] [12].

Con el pasar del tiempo, la IA ha sido empleada con mayor frecuencia en lugar de los métodos tradicionales para resolver problemas. Lo anterior es posible gracias a la investigación y desarrollo de algoritmos, con lo cual, las técnicas de IA son cada vez más sofisticadas a la par de producir resultados satisfactorios para el usuario [5]. De manera esencial, se utilizan aplicaciones de IA para resolver problemas de manera “inteligente” escogiendo la mejor decisión en un vasto universo de posibilidades [1] [2].

Si se rastrea el origen de la expresión “inteligencia artificial” es menester mencionar el documento de *“Dartmouth Summer Research Project on Artificial Intelligence”* publicado en el año 1955, como una de las primeras referencias al termino. J. McCarthy junto con Marvin Minsky, Nat Rochester y Claude Shannon, sugieren el termino para referirse al diseño de máquinas con la capacidad de realizar acciones humanas que habitualmente son catalogadas como inteligentes [4].

Una definición más actualizada en donde se abarcan más aspectos envueltos en la inteligencia artificial es proporcionada por la Universidad de Oxford: *“La teoría y desarrollo de sistemas computacionales capaces de realizar tareas que normalmente requerirían inteligencia humana como la percepción visual, reconocimiento del habla, toma de decisiones y traducción entre idiomas”* [19].

Por lo que la inteligencia artificial, es una de las ramas dentro de las ciencias computaciones encargada de estudiar, generar y proponer algoritmos con la capacidad de resolver tareas y problemas de manera que pueda ser calificada como inteligente [4] [19].

Gracias al constante progreso y el desarrollo en el área de la inteligencia artificial, ha sido posible generar nuevas líneas de investigación como lo son: sistemas de partículas [5], computación evolutiva [6], agentes inteligentes [7], redes neuronales [5], algoritmos de optimización [2], por mencionar algunos ejemplos [1].

Desde los inicios de la inteligencia artificial, los problemas de optimización combinatoria han sido estudiados como campos de aplicación para algoritmos y heurísticas [20]. Debido a que ofrecen un amplio rango de posibilidades, las cuales se encuentras delimitadas por un conjunto de restricciones, por lo que diseñar estrategias que puedan proveer soluciones no es una tarea trivial [2].

Cuando los problemas de optimización combinatoria son sencillos; es decir, donde es posible revisar el abanico de posibilidades en un tiempo razonable (usualmente definido en términos de minutos), es factible emplear estrategias de fuerza bruta [12].

Sin embargo, en escenarios donde explorar cada posible ramificación del problema, es una tarea infactible en término de tiempo, debido a que podría tomar días, meses e incluso años, es cuando se debe considerar el uso de una técnica de inteligencia artificial [20].

Una gran parte de los problemas que se presentan en el mundo real, requieren diferentes objetivos que deben ser maximizados (ganancias, satisfacción, beneficio, etc) o minimizados (tiempo, costo, perdidas, etc). Sin embargo, a la par de esos recursos que deben ser maximizados, existen también una serie de limitantes que deben ser respetadas para que una solución se considere como valida. Lo anterior descrito puede ser catalogado como un problema de optimización [12].

2.1 Optimización

La optimización tiene la intención de realizar una maximización (o minimización dependiendo del caso) de beneficios con recursos limitados. Por lo que objetivo final de la optimización es construir una solución que proporcione los mejores resultados para el problema [12].

Se podría hacer una analogía como la empleada por el autor Yang [21], en donde una persona se encuentra en un bosque, sabe de la existencia de diamantes, pero no conoce la ubicación exacta. El encontrar las gemas es el objetivo primordial, el cómo realizarlo es el problema [21].

Buscar centímetro a centímetro, nos generaría un enorme consumo de energía y tiempo. Por lo que una mejor aproximación sería obtener información sobre puntos potenciales de gemas, de tal forma que nuestra energía sea concentrada en áreas prometedoras. De esta manera se realiza una búsqueda inteligente, menos laboriosa y en menor tiempo.[12]

De acuerdo a [22] [23] un problema de optimización pude definirse de la siguiente manera:

$$P = (S, \Omega, f)$$

Donde P es un problema de optimización, S es el espacio de búsqueda el cual está definido en base a las variables de decisión $X_i, i = 1, \ldots n, \Omega$ es el conjunto de restricciones que están relacionadas con las variables de decisión y finalmente f es una función objetivo que se desea optimizar [22] [23].

Para resolver P se debe buscar una solución s_0 en S de tal manera que cumpla: $f(s_0) \leq f(s), \forall s \in S$ para minimización o $f(s_0) \geq f(s), \forall s \in S$ en caso de ser maximización [22].

Un espacio de búsqueda es el conjunto de todas las posibles soluciones para un problema de optimización. Debido a que el espacio solución viable para un problema está acotada por las restricciones, no todas las soluciones serán válidas [22] [23].

Las variables de decisión, son aquellas características que tienen un efecto sobre el problema. No todas las variables impactan de la misma manera al problema, pero es menester elegir con cuál de las variables se va a realizar la optimización [32] [23].

Muchos de los problemas de la vida real tienen limitaciones que deben ser respetadas (cantidad de recursos, tiempos de entrega, costos, por mencionar algunos), en los cuales rebasar esas limitaciones no es una opción. Por lo que las restricciones vienen a representar ese tipo de limitaciones a las que están sujetos los problemas [32] [23].

Como se ha descrito anteriormente, todo problema de optimización contiene las siguientes características: variables de decisión, una función objetivo que necesita ser maximizada u minimizada, además de una serie de restricciones las cuales suelen estar en conflicto unas con otras [22] [23].

Por lo que las técnicas de optimización tienen el objetivo de encontrar los valores para las variables de decisión, con la intención de generar el mejor resultado posible en la función objetivo, a la par de respetar las restricciones impuestas [22] [23].

2.2 Heurística

La creación de algoritmos exactos para la resolución de problemas particulares, ha sido posible debido a que las propiedades de las tareas a resolver, han sido comprendidas a tal profundidad, que es factible diseñar una serie de pasos que permiten llegar a la solución óptima [1]. Sin embargo, existen problemas para los cuales aún no se ha encontrado un esquema determinista que lo resuelva, es aquí donde se abre un espacio para el uso de las heurísticas como estrategia para abordarlos [2] [5] [23].

Una heurística es una técnica de las ciencias computaciones cuyo propósito es resolver un problema en el que pueden emplear mecanismos estocásticos o aleatorios como uno de sus componentes esenciales [1]. La exploración del espacio solución que realizan las heurísticas no es de manera exhaustiva, lo que permite tener una noción general del problema y concentrarse en áreas prometedoras [1] [2].

Los esquemas heurísticos, tienen propiedades ventajosas con respecto a los algoritmos deterministas: encuentran una solución factible (no necesariamente la mejor) en tiempo polinomial, son diseñados de manera particular para atacar a la tarea, emplean elementos configurables y de aleatoriedad que permiten obtener soluciones distintas en caso de que un primer resultado no sea el esperado [1] [2] [5].

Por lo descrito en el renglón anterior, un método heurístico no se tiene la certeza de obtener la solución óptima del problema. Esta deficiencia es compensada por un costo de tiempo bajo en ejecución (definido en término de minutos), a la par de encontrar soluciones factibles para el problema [2] [5] [23].

Para lograr concluir con una solución factible, el algoritmo heurístico debe comenzar con una solución inicial (que puede ser creada de manera aleatoria o construida siguiendo una estrategia), para después ser modificada por una serie de mecanismos internos, para paulatinamente mejorar la calidad de la propuesta inicial [1] [2] [5].

Debido a lo anterior, las heurísticas tienen una propiedad de interés para los usuarios y es que, tras recibir una misma instancia de entrada, se pueden obtener soluciones distintas en cada ejecución que se realiza, con lo cual sí un investigador no está satisfecho con una solución resultante, puede volver a iniciar el programa esperando una nueva propuesta que cumpla con sus expectativas [1] [5].

2.3 Metaheurísticas

Gran parte de los problemas del mundo real son altamente complejos de resolver, debido a que manejan un gran número de variables que pueden ser: discretas, continuas, calificativas o una mezcla de todas las anteriores. Sin mencionar que usualmente, se encuentran restringidos por limitaciones (e.g. tiempos de entrega, recursos limitados, costos, etc.) [2] [22].

Hablando particularmente de los problemas de optimización combinatoria, por lo general, son problemas los cuales poseen una inmensa cantidad de posibles respuestas factibles para resolver la tarea. Sin embargo, el usuario sólo quiere la mejor solución, por lo que encontrar el candidato que proporcione los beneficios más altos, no es una labor trivial [22].

Una posible forma de abordar esta clase de problemas podría ser utilizar un algoritmo de fuerza bruta. Un algoritmo de fuerza bruta tiene la propiedad de generar cada una de las posibles soluciones, de tal forma que sí se califica cada uno de los candidatos, en algún momento se encontrará la respuesta que proporcione los mejores resultados [1] [5] [23].

Hay que tener claro que evaluar cada posible candidato, requiere de una determinada cantidad de tiempo y es ahí donde reside el mayor defecto de las estrategias de fuerza bruta. Para problemas que puedan ser resueltos en cuestión de minutos, es viable emplear este tipo de técnicas [2] [21] [22].

Sin embargo, para tareas donde revisar cada posible solución nos lleve una cantidad de tiempo del que no se está dispuesto a esperar (lo anterior en termino de días, meses e inclusive años), se debe de optar por otro tipo de estrategia [1] [21] [24].

Cuando los problemas son costosos de resolver en términos de tiempo, memoria o procesamiento empleando algoritmos exactos, es donde se abre la posibilidad de emplear estrategias de tipo metaheurístico para afrontar estas tareas [1] [21].

De manera opuesta a los métodos exactos, los cuales encuentran una solución a expensas de emplear grandes cantidades de recursos computacionales, los métodos metaheurísticos encuentran soluciones aproximadas al óptimo empleando menos recursos de computación [23].

Sí se analiza la palabra *metaheurística*, se encuentra en su composición el prefijo griego meta además de la palabra *heurística*, por lo que se podría definir como: *algoritmos heurísticos superiores o de un nivel más alto* [11]. A diferencia de las heurísticas, las metaheurísticas buscan ser esquemas de uso general, es decir;

una metaheurística puede ser empleada en una mayor cantidad de problemas, modificando sus componentes para adaptarse al campo de búsqueda. Lo anterior descrito es posible gracias a que integran en su diseño componentes que les permiten ampliar sus rangos de búsqueda, así como mecanismos con la capacidad de refinar los resultados encontrados [11] [22] [23].

Las metaheurísticas, al igual que las heurísticas, pueden contener mecanismos estocásticos o de aleatoriedad, que permiten obtener soluciones distintas cada vez que son ejecutados, lo que los convierte en algoritmos no deterministas [25] [26].

Son gracias a los mecanismos de aleatoriedad, lo que dota a las metaheurísticas de la capacidad de exploración del espacio solución de un problema. Si la estrategia es adecuada para el problema, tras terminar su ejecución obtendrán soluciones adecuadas o cercanas al óptimo (en caso de conocerlo) de manera consistente [27] [28].

Son gracias a las propiedades de: obtener soluciones adecuadas o cercanas al óptimo, sumado a encontrar respuestas en tiempos aceptables (donde las ejecuciones terminan en minutos) y de poseer componentes modificables para diferentes problemas, lo que ha vuelto a las estrategias metaheurísticas exitosas como herramienta para abordar tareas de optimización [29] [30].

Cabe señalar que adaptar una metaheurística a un problema de interés no es una labor fácil [27]. El investigador gasta gran cantidad de tiempo en la elección de los componentes, así como también la estructura de la estrategia para el control de flujo de la información, es decir: el orden en el que los componentes de la metaheurística van a ser ejecutados [28] [29].

Además, cada mecanismo tiene una serie de parámetros asociados para su funcionamiento [27]. El modificar cada uno de parámetros afecta de manera directa

el funcionamiento del componente, por ende, también se ven afectados los resultados obtenidos por el algoritmo [29] [31].

Por lo descrito en los párrafos anteriores, es menester mencionar alguna de las propiedades más importantes de una metaheurística, así como los tipos de búsqueda que realiza y una descripción de los enfoques más empleados para comprender de manera clara el funcionamiento de la técnica [23] [27].

2.3.1 Explotación y exploración

Existen dos características que pueden ser consideradas como las piedras angulares para resolver un problema de optimización de manera exitosa y consistente en cada ejecución que realice el algoritmo; la exploración y la explotación [23] [27].

La exploración y explotación (también referidos como diversificación e intensificación, divergencia y convergencia, respectivamente) son características que cualquier metaheurística debe contener en sus mecanismos internos [2]. Si bien se busca un equilibrio de ambos elementos, es posible que ciertas estrategias se inclinen más hacía un aspecto, pero no dejan de contemplar ambos [32][33].

La exploración es la propiedad de expandir el rango de búsqueda en un problema [23]. La intención principal es tener un espectro más amplio de las regiones del espacio solución del problema, con la finalidad de tener una posibilidad mayor de encontrar regiones prometedoras [34] [35].

Mientras que la explotación se realiza una vez se tenga la información preliminar del problema, para concentrar los esfuerzos del programa en indagar en áreas potenciales, concluyendo con una solución óptima [23] [27].

De manera usual, un algoritmo para tener características de exploración, suele construir las soluciones iniciales de manera aleatoria [23]. De tal manera que, en cada nueva ejecución o generación de solución, la técnica comience en un punto distinto [32]. Además de implementar mecanismos que generen cambios significativos en la estructura del candidato, con la intención de dotar a la estrategia la capacidad de explorar diferentes regiones a la original [36].

Por otro lado, también se suele integrar un método de búsqueda intensiva que trabaje con la información obtenida por los mecanismos de exploración [32]. Para intensificar la explotación, el algoritmo contiene mecanismos que realizan cambios mínimos a la estructura de la solución, con la finalidad de generar una convergencia hacía un punto [23] [35].

2.3.2 Enfoque de solución única y basado en población

Como se ha mencionado, cuando se emplean las estrategias metaheurísticas para resolver un problema de optimización, se espera obtener una solución que pueda ser de utilidad para el usuario. Para llegar a una respuesta, las técnicas metaheurísticas dentro de su diseño contemplan el número de candidatos a utilizar en su proceso de búsqueda [2].

De acuerdo al número de soluciones a emplear en el proceso de búsqueda, las metaheurísticas pueden ser catalogadas en dos grupos: las estrategias que emplean una sola solución [37] y las técnicas que utilizan un grupo de soluciones [38]. Esta propiedad es muy importante al momento de elegir un algoritmo, debido a que cada enfoque posee una serie de ventajas y desventajas intrínsecas en su planteamiento [2].

Para el enfoque donde se emplea una única solución, solamente puede existir un único candidato con el que se esté trabajando en todo el proceso de

optimización [2]. Para efectuar la búsqueda, usualmente los algoritmos comienzan con cambios bruscos en la respuesta, con la intención de ir explorando las regiones del espacio solución [30]. Conforme pase el tiempo, los cambios proceden a ser mínimos para refinar al candidato [35].

Cabe indicar que, durante las etapas de búsqueda, puede existir la posibilidad de perder una buena solución (inclusive la mejor encontrada), por lo que se suele implementar un mecanismo que guarde el mejor candidato hallado [30] [35].

Hay que destacar que los enfoques de solución única consumen pocos recursos de memoria, además de que pueden realizar un mayor número de operaciones en el mismo tiempo a las que podría realizar las estrategias que emplean un grupo de candidatos, debido a que sólo deben de aplicar sus mecanismos a un único individuo [2] [30].

Debido a que solo se emplea una única solución, el candidato se ve constantemente en proceso de refinamiento de sus características durante el proceso de búsqueda, por lo que la característica de explotación se ve altamente beneficiada de este enfoque [30][35].

Para el enfoque donde se emplea un grupo de candidatos (también denominado población), el algoritmo trabaja con todo el conjunto de soluciones empleadas en proceso de optimización [6]. Durante cada etapa del algoritmo, la población debe ser alterada con el objetivo de expandir el rango de exploración e ir acumulando conocimiento de las regiones visitadas [29] [38].

Por lo general, este tipo de algoritmos emplean mecanismo que tienen el objetivo de manipular un subgrupo de soluciones con el propósito de: generar múltiples combinaciones entre individuos, modificar propiedades de los candidatos

y refinar las soluciones. Por lo cual, son los métodos internos los que proporcionan las características de exploración y exploración [6] [38].

Cabe mencionar que los enfoques de población consumen un mayor número de recursos de memoria y procesamiento en el equipo de cómputo [38]. Debido a que cada mecanismo de la metaheurística debe ser aplicado a los individuos de la población, pero es compensado con la ventaja de explorar una mayor región del espacio solución [6].

2.3.3 Función objetivo

Durante el proceso de optimización, es necesario generar constantemente propuestas, por lo que se vuelve necesario un mecanismo que tenga la capacidad de calificar cada nueva respuesta construida [2]. La tarea se vuelve crucial para establecer si los cambios ejecutados han mejorado la solución con respecto a la inicial, o si por el contrario ha empeorado [33] [36].

Establecer de manera cuantitativa un valor a cada individuo, se vuelve una tarea esencial para el funcionamiento del algoritmo [6]. Por lo que, en el diseño de la metaheurística, se debe contemplar un mecanismo que cubra tales fines [2]. De manera común, el método que se utiliza es el denominado función objetivo [30] [35].

De manera usual, las funciones objetivo tienen en su composición las variables de decisión del problema de optimización [36]. La importancia con la que cada variable afecta a la función objetivo, habitualmente lo define el problema de estudio o el investigador [34]. Por lo que no hay una única forma de definir una función objetivo y cada cambio realizado a la estructura de la misma, puede impactar en el desempeño de la metaheurística [30] [35].

Como se ha mencionado, proponer la función objetivo depende del problema de estudio y de lo que el investigador desea evaluar, pero particularmente para los problemas de optimización, es preferible que pueda ser planteada en términos de objetivos que puedan ser maximizados o minimizados [32] [33].

2.3.4 Condición de paro

Es necesario especificar una condición o situación para detener el proceso de optimización y devolver la mejor solución encontrada, de lo contrario el algoritmo se encontraría en un ciclo perpetuo. Las condiciones de finalización más comunes para una metaheurística son: número de iteraciones, valor esperado y tiempo de ejecución [37]. La elección de cada uno varía de acuerdo al problema y es definido por el investigador [30] [35].

Establecer un número de iteraciones es para algunos casos de estudio, suficiente para encontrar soluciones adecuadas en un tiempo aceptable [30]. El encontrar la cifra ideal es una ardua tarea, puesto que cada instancia es diferente y cada caso de estudio es distinto. Con lo cual, el valor seleccionado se obtiene con base a experimentos y resultados obtenidos en distintos casos de prueba [35].

Utilizar una solución estimada, tiene el beneficio de que la metaheurística seguirá trabajando hasta encontrar una solución con un valor esperado establecido por el investigador. Sin embargo, utilizar este criterio corre el riesgo de nunca terminar, ya sea porque no se ha encontrado la solución que cumpla los requisitos (e.g. el algoritmo se ha estancado en su búsqueda) o simplemente no existe candidato que cumpla las condiciones exigidas [39][40].

Establecer un tiempo de ejecución es un método que garantiza un uso adecuado de los recursos computacionales [36][41]. El umbral es establecido por el usuario, y va acorde a los recursos de los que se dispone [35]. En este caso, se

corre el riesgo de que el proceso de optimización necesite de un mayor periodo de tiempo, lo que desemboca en una solución que aún podría ser mejorada [42] [43].

Capítulo 3

3. Hiper-heurísticas

Dentro del campo de los problemas de combinatoria, las metaheurísticas han resultado ser herramientas poderosas para construir soluciones aproximadas al óptimo [1]. En general, las metaheurísticas puede obtener resultados aceptables sin requerir la integración de componentes particulares que mejoren su rendimiento ante un problema de estudio [2]. Sin embargo, las técnicas más eficientes para resolver una sola clase de problema, han sido diseñados específicamente para trabajar sobre esa tarea [10] [11].

Esto implica que existe un alto grado de conocimiento del investigador tanto del problema, sus detalles distintivos, las relaciones intrínsecas entre cada una de las variables, así como las técnicas computacionales existentes, sus características, sus ventajas y desventajas. [10] [11].

Por lo que si se agrega una instancia desconocida al problema puede significar: rediseñar, integrar, intercambiar o mejorar un mecanismo de la metaheurística inicial, para que resuelva la nueva instancia del problema con los mismos resultados, lo cual es una labor que requiere de tiempo [10] [11].

Adicionalmente, las metaheurísticas suelen tener una serie de diferentes parámetros para su funcionamiento [2]. Cada uno de esos valores, debe ser ajustado de manera minuciosa al problema de estudio o incluso a una instancia particular del mismo [10] [11].

Por lo que agregar una instancia desconocida, podría impactar en un reajuste de parámetros del algoritmo para abordar de manera eficiente el problema [10]. Lo anterior surge, debido a que las metaheurísticas realizan su búsqueda sobre el espacio solución del problema, es decir trabajan directamente con el problema y la respuesta (o respuestas) que están refinando [9] [11].

Finalmente, si el algoritmo propuesto quisiera ser probado en otro problema de un dominio diferente, grandes cambios deben ser realizados a la estructura del algoritmo, así como a los mecanismos y parámetros de la estrategia [15].

Es en este tipo de escenarios es donde se abre la apertura para el uso de una hiper-heurística [46]. Los algoritmos hiper-heurísticos han emergido en un intento de proporcionar un mayor nivel de generalización a la obtención de soluciones para distintos problemas de optimización combinatoria [47] [48]. Adicionalmente, las estrategias hiper-heurísticas buscan obtener un desempeño competitivo en distintos problemas de dominio, en lugar de producir resultados sobresalientes para un sólo tipo de problema [15].

A diferencia de las estrategias metaheurísticas, una hiper-heurística es una estrategia que plantea la búsqueda sobre un espacio heurístico, lo que provee a la técnica de un mayor nivel de generalización, a diferencia del espacio solución donde buscan las metaheurísticas [48] [49]. Es decir, las hiper-heurística trabajan de tal manera que son capaces de elegir a una heurística apropiada para el problema que se le presente [1] [9] [46].

Como se mencionó, una hiper-heurísticas buscan sobre el espacio heurístico, para lograrlo, el algoritmo se integra de dos componentes principales: el nivel superior y el nivel inferior [47]. En el nivel superior puede operar una técnica metaheurística o cualquier otro mecanismo que guía el proceso de búsqueda [9]. En el nivel bajo se encuentran las heurísticas (también llamadas *heurísticas de bajo nivel*) que trabajan directamente con el problema [10] [11] [49].

De manera tal que el nivel más alto tiene la labor de buscar sobre el espacio heurístico (el nivel bajo) y el nivel inferior explora sobre el espacio solución [9]. Con la finalidad de lograr un nuevo grado de generalización, reflejado en la capacidad de abordar una mayor cantidad de problemas con poca o nula integración de nuevos componentes [10] [11].

Cabe señalar que existe un último componente entre el nivel superior y el nivel inferior de la hiper-heurística denominado “barrera de dominio”. La barrera de dominio es el mecanismo que separa ambos niveles, así como también realiza el intercambio de información entre los niveles [9] [10] [11].

Es importante notar que las hiper-heurísticas comparten la misma deficiencia que las metaheurísticas: requieren de configurar una serie de parámetros cruciales para su funcionamiento adecuado, la cual no es una tarea trivial [1]. El ajuste de los indicadores requiere de un conocimiento profundo del funcionamiento del programa, además de una cantidad considera de tiempo para pruebas [2] [47].

3.1 Nivel alto y nivel bajo

El esquema clásico de las hiper-heurísticas contempla dos componentes esenciales en su estructura: el nivel alto y el nivel bajo [9]. El nivel alto, permite al algoritmo dirigir el proceso de búsqueda sobre el espacio heurístico [47] [48]. Mientras que el nivel inferior contiene a las heurísticas de bajo nivel, las cuales trabajan directamente con el problema [49] [50].

De manera habitual, tanto el nivel alto como el nivel bajo tienen enfoques frecuentes para su funcionamiento [9]. En el nivel superior existen dos ramas

típicas: esquema por selección y elección. Mientras que, en el nivel inferior, se contemplan las opciones de: perturbación y construcción [10] [11] [47].

Esencialmente, el enfoque de selección tiene la intención de elegir una heurística de bajo nivel que sea adecuada para una determinada fase del problema [45]. Mientras que el esquema de generación pretende a partir de las heurísticas de bajo nivel (las cuales pueden ser sólo componentes) construir una heurística que sea capaz de adaptarse al problema que se está resolviendo [46].

Continuando con el nivel inferior, las heurísticas de perturbación de bajo nivel, son técnicas que realizan modificaciones a una solución inicial (ya sea generada de manera aleatoria o construida siguiendo un tipo de estrategia) [50]. La finalidad de las perturbaciones es desplazarse en el vecindario que posee el candidato [9]. La forma de realizar los cambios depende altamente del problema a resolver, por lo que son mecanismos dependientes del problema [51] [52].

Las heurísticas de construcción de bajo nivel, son técnicas que tienen la intención de generar una solución inicial con las heurísticas de nivel inferior que poseen. La respuesta generada puede ser usada después como punto de partida para otras técnicas de optimización como gran diluvio, recocido simulado o búsqueda tabú [9] [46]. De manera similar que el enfoque de perturbación, las estrategias de construcción de bajo nivel tienen conocimiento del problema [51] [52].

Finalmente, de acuerdo a la elección de los enfoques del nivel alto y del nivel bajo de la hiper-heurística es posible realizar una clasificación [10] [11]. Si en el diseño del algoritmo, el nivel alto es de tipo selección, entonces se pueden hacer dos ramificaciones: hiper-heurística de selección-perturbación (HHSP) e hiper-heurística de selección-constructiva (HHSC) [9][46]. Por otro lado, si el nivel superior se decantó por un enfoque de generación, entonces de manera similar se

puede tener dos estructuras principales: hiper-heurística generación-perturbación (HHGP) e hiper-heurística generación-construcción (HHGC) [9] [46].

3.2 Clasificación de enfoque hiper-heurísticos

De acuerdo a los enfoques del nivel superior e inferior de la hiper-heurística es posible realizar una clasificación de las hiper-heurísticas [10] [11]. Para el nivel superior hay dos vertientes principales: esquema por selección y por generación, a su vez, dependiendo del enfoque inferior se puede tener otras dos variantes: enfoque de perturbación y enfoque de construcción [9].

En base a lo descrito en el párrafo anterior, es posible tener hasta cuatro tipos de estrategias: hiper-heurística de selección-perturbación (HHSP), hiper-heurística de selección-constructiva (HHSC), hiper-heurística generación-perturbación (HHGP) e hiper-heurística generación-construcción (HHGC) [9].

La clasificación anterior puede exponer de manera clara los posibles diseños de una hiper-heurística, aunque existen propuestas que no están contempladas en la clasificación. Por ejemplo, un nivel bajo donde se combine tanto estrategias de perturbación y heurísticas de construcción. Pero para propósitos de ilustración, basta con la clasificación realizada [9] [12].

3.2.1 Hiper-heurística selección-constructiva

En este tipo de enfoque, se tiene el objetivo de generar una solución por medio de selección y aplicación de las heurísticas de bajo nivel. La solución construida, después puede ser utilizada como punto de partida para otros algoritmos de optimización [9] [37].

Para lograrlo, las HHSC primero comienzan con una solución vacía, después aplican una heurística de nivel inferior, lo que permite pasar a un nuevo estado en la solución, se repite los pasos de selección y aplicación hasta tener una solución final. De tal manera que la solución improvisada, fue elaborada utilizando las herramientas de bajo nivel disponibles [10] [37].

De manera similar a HHSP, las heurísticas de bajo nivel de HHSC son altamente dependientes del problema de estudio. En cuanto al nivel alto, se pueden emplear diversas estrategias para su funcionamiento: razonamiento caso base, métodos de búsqueda local, métodos basados en población por mencionar algunos [12].

Es importante destacar que las soluciones construidas por este enfoque deben de ser factibles, lo que vuelve más complejo el diseño de HHSC y por ende se debe contemplar un mecanismo para tales propósitos [12].

Para finalizar las características de HHSC, se muestra un pseudocódigo del funcionamiento [9].

```
HHSP:
    s = solucion_inicial_vacia()
    MIENTRAS: solucion_incompleta(); HACER:
        h = seleccionar_heuristica(H)
            s = h(s)      // aplicar la heurística h a la solución s
                          // para extender s
    REGRESA s
```

3.2.2 Hiper-heurística de selección perturbación

Una hiper-heurística de selección-perturbación tiene el enfoque de elegir una heurística de bajo nivel, con la intención de ser aplicada en algún punto del

proceso de búsqueda a una solución recibida. Cabe destacar que la solución entregada puede ser aceptada o rechazada [47].

La forma en la que ejecuta la exploración la hiper-heurística de selección-perturbación, es por medio de modificaciones a la solución con la que se está trabajando [51]. Aplicando las heurísticas de bajo nivel, el algoritmo puede desplazarse por los vecindarios que posee la solución [9].

La manera típica de trabajar de las HHSP, es aplicar iterativamente las heurísticas de perturbación hasta llegar a un estado en el que no existe posible mejora en la solución actual [9] [12]. Para determinar cuando no hay obtención de beneficio en la solución, el algoritmo emplea algún tipo de medición cuantificable como puede ser el valor generado por una función objetivo [50].

Cabe señalar que las heurísticas de perturbación de bajo nivel son dependientes del problema, es decir, las heurísticas deben tener conocimiento de la tarea que realizan [9]. Por ejemplo, para un problema de horarios, una propuesta de bajo nivel podría intercambiar dos eventos en el calendario [52].

Además de sus dos niveles básicos, una hiper-heurística de selección-perturbación contempla en su diseño un componente para determinar si el cambio realizado en una solución es aceptado o rechazado [47]. El método a emplear no tiene algún tipo de limitación en su diseño, por lo que hay flexibilidad de elección para el investigador [12]

Cabe señalar que las HHSP, pueden realizar las perturbaciones a una sola solución o múltiples soluciones [9]. Para el enfoque de múltiples soluciones, las hiper-heurísticas se apoyan de técnicas poblacionales (como los algoritmos genéticos) para su funcionamiento [12] [38].

Para finalizar las características de HHSP, se muestra un pseudocódigo del funcionamiento [9].

```
HHSP:
    s = generar_solucion_inicial()
    MIENTRAS: criterio_finalizacion(); HACER
        h = seleccionar_heuristica(H)
        nueva_s = h(s)      // aplicar la heurística h a la solución s
        SI criterio_aceptacion(nueva_s):
            s = nueva_s  // se actualiza s
    REGRESAR s
```

3.2.3 Hiper-heurística de generación-construcción

En las hiper-heurísticas de generación-construcción, se pretende elaborar una solución inicial de manera incremental [9]. Para lo cual, primero partiendo de las heurísticas o componentes heurísticos del nivel inferior, se genera una nueva heurística [10] [11]. La nueva heurística después construye una solución inicial que luego será utilizada como punto de partida por otra técnica de optimización [26].

En las aproximaciones hiper-heurísticas HHSP y HHSC, las heurísticas del nivel bajo son dependientes del problema y son hechas a la medida del problema, además de que son diseñadas manualmente por el intelecto humano, por lo que proponer heurísticas de bajo nivel tiene un costo alto de tiempo asociado [9]. Las HHGC tienen como principal aportación la automatización del proceso anterior [9] [12].

Automatizar el proceso de proponer heurísticas de bajo nivel, en principio, reduce las horas de dedicadas al diseño, además, de que por medio de HHGC se pueden probar un mayor número de heurísticas de las que un humano podría, e inclusive llevar a estrategias que una persona no hubiera pensado en un principio [9] [10] [12].

Esto dota a las HHGC de dos características importantes: la primera es que puede crear heurísticas de bajo nivel para una instancia particular a partir de los componentes de los que dispone, y a la vez que tenga un mayor grado de adaptabilidad para instancias no vistas e inclusive para diferentes clases de problemas [11] [12].

Por último, cabe señalar que las heurísticas generadas pueden ser almacenas, para luego ser aplicadas a otras instancias no vistas del problema, adicional a lo anterior, se puede tener un conjunto de heurísticas iniciales para abordar un problema desconocido [9] [12].

3.2.4 Hiper-heurística generación-perturbación

En las hiper-heurísticas de generación-perturbación, se pretende crear una nueva heurística de perturbación de bajo nivel [9]. Lo anterior es logrado mezclando componentes heurísticos existentes en el nivel inferior, con la intención de generar una nueva estrategia [12].

La manera usual del funcionamiento de la hiper-heurística de generación-perturbación es el siguiente: a partir de una solución inicial (que ha sido generada de manera aleatoria o por otro algoritmo), HHGP intenta crear una heurística de bajo nivel que pueda realizar modificaciones a la solución, con la intención de generar un beneficio [9] [46].

Cabe señalar que la heurística de perturbación creada, es dependiente del problema y a menudo, los alcances que tiene la heurística generada son sólo de búsqueda local [12]. Lo anterior debido a que la forma de generar la estrategia es a partir de componentes de bajo nivel, los cuales tienen conocimiento limitado acerca del funcionamiento del problema [9][46].

De manera similar a HHGC, las heurísticas de bajo nivel generadas por HHGP puede ser almacenas para luego ser aplicadas a instancias no vistas de un problema, lo que, en principio, dota de diversas herramientas para abordar al problema de estudio, y reduce las horas de diseño por parte del investigador para proponer diferentes esquemas de perturbación [9] [12].

3.3 Aprendizaje en las hiper-heurísticas

El aprendizaje en las hiper-heurísticas es algo que puede ser implementado. En la estrategia se contemplan tres posibles escenarios de aprendizaje: en línea (*online*), fuera de línea (*offline)* y nulo. Cabe señalar que el aprendizaje online y offline pueden estar presentes en un mismo algoritmo [12].

El aprendizaje en línea se presenta cuando hay una retroalimentación por parte del problema y hay modificaciones en las configuraciones iniciales del algoritmo [14] [38]. El aprendizaje fuera de línea ocurre sí la hiper-heurística ha guardado conocimiento previo adquirido al resolver otras instancias del problema, con la intención de ser aplicado a ejemplos desconocidos [10] [11]. Finalmente, sí no hay cambio alguno en la hiper-heurística, entonces hay un aprendizaje nulo [12] [46].

Las heurísticas que no utilizan ningún tipo de mecanismo de aprendizaje se basan en una selección aleatoria o en una búsqueda exhaustiva. Por lo que el desempeño del algoritmo depende de la configuración inicial dada por el investigador. [12]

Las hiper-heurística que presentan aprendizaje en línea, se encuentran basados en la implementación de mecanismos que tengan la capacidad de calificar un determinado aspecto del algoritmo [14] [38]. Típicamente evaluar el desempeño

de las heurísticas de bajo nivel durante cada etapa del proceso de optimización [51] [52].

Usualmente los mecanismos de aprendizaje en línea suelen presentar las siguientes características: una puntuación inicial, una memoria definida, una estrategia base para la selección (con base en la puntuación) y un mecanismo de actualización de los puntajes obtenidos durante cierto periodo de tiempo [14] [51].

La puntuación inicial es la calificación con la que comienza la hiper-heurística, el tamaño de la memoria determina el impacto del desempeño durante un periodo de tiempo, la estrategia de elección guía el proceso de búsqueda en la técnica y el mecanismo de actualización tiene la finalidad de refinar el rendimiento general del algoritmo [38] [52].

Después de resolver un problema y que la hiper-heurística haya generado un conocimiento, sería menester implementar un mecanismo que guarde el aprendizaje realizado [46]. Es este tipo de escenarios es idóneo la integración de métodos de aprendizaje fuera de línea [12].

El aprendizaje fuera de línea dota de dos características importantes a las hiper-heurísticas: la primera es que el proceso de aprendizaje realizado no tenga que ser repetido nuevamente, además de que el conocimiento puede seguir siendo acumulado con la intención de mejorar el desempeño para una nueva instancia. Cabe señalar que el desempeño del algoritmo está ligado al conjunto de pruebas con el que fue entrenado [12] [46].

3.4 El desafío heurístico de búsqueda de dominio cruzado y HyFlex

Uno de las finalidades que persiguen las hiper-heurísticas es: proporcionar estrategias de un nivel de mayor generalidad, que produzca buenos resultados para una serie de problemas de dominio, en lugar generar resultados sobresalientes para uno o dos casos particulares y respuestas deficientes para las otras [53]. Es aquí en donde el *desafío heurístico de búsqueda de domino cruzado* (del inglés Cross-Domain Heuristic Search Challenge) busca lograr el objetivo anterior [9] [15] [53].

Las hiper-heurísticas presentadas para el desafío heurístico de búsqueda de dominio cruzado (también conocido en la literatura como CHeSC) buscan producir resultados competitivos a una serie de problemas de dominio, en lugar de obtener un resultado sobresaliente para una instancia de un problema y generar resultados pobres en las otras tareas [12] [53].

La propuesta de CHeSC se ha concentrado particularmente en problemas discretos de optimización combinatoria [12]. Donde las hiper-heurísticas de selección-perturbación han tenido una participación destacada en el reto [11].

CHeSC fue una competencia realizada en el año 2011 enfocada a la comunidad que investiga a las hiper-heurísticas, con la finalidad de proponer una estrategia que obtuvieran resultados competitivos en un conjunto de seis problemas distintos de optimización discreta combinatoria [9] [15]. Si bien, han existido hiper-heurísticas que han generado resultados aceptables en uno o dos problemas de dominio, no se había realizado un esfuerzo en evaluar a los algoritmos en una mayor cantidad de problemas de dominio [53].

Si un investigador quisiera participar en CheSC, se volvería complicado para el desarrollador adaptar una propuesta de algoritmo a seis problemas distintos, por

no mencionar el tiempo que implicaría tal labor [12]. Por tal motivo, se volvió indispensable el programar una herramienta que proporcione una interfaz entre los desarrolladores y los problemas [53].

De esta manera el algoritmo planteado no necesita conocer a detalle cada problema de dominio, con lo cual el investigador puede concentrarse en el diseño de la hiper-heurística [9]. Por lo anterior fue necesario programar la herramienta de HyFlex [53].

HyFlex (abreviatura de: Hyper-heuristic Flexible framework), es una herramienta de desarrollo de software enfocada al diseño de hiper-heurísticas y metodologías para los problemas del dominio cruzado [53]. La herramienta es una librería que contiene métodos de desarrollo e interfaces, con las cuales un algoritmo puede abordar diferentes problemas de optimización combinatoria sin cambiar su estructura [9] [12] [53].

La librería HyFlex fue desarrollada y entregada a cada uno de los participantes del desafío [12]. Con la cual, los concursantes podían implementar una hiper-heurística que pudiera ser aplicada a los seis problemas sin realizar cambios en la estructura de la propuesta [9]. En el año 2014 CheSC fue extendida permitiendo las estrategias que emplearan multi-hilos en su funcionamiento. En cuanto a la librería, los competidores contaban con las siguientes herramientas [12] [53]:

- Métodos para crear una solución inicial.
- Métodos para calcular la aptitud de un individuo.
- Heurística de perturbación de bajo nivel.
- Instancias del problema de dominio.

La librería implementa lo anterior para la siguiente serie de problemas de optimización combinatoria [9] [12]:

- Satisfacción booleana (del inglés: *Boolean satisfiability*)
- Embalaje de un contenedor unidimensional (del inglés: *One-dimensional bin packing*)
- Permutación para flujo de tienda (del inglés: *Permutation flow shop*)
- Programación de horarios para personal (del inglés: *Personnel scheduling*)
- Vendedor ambulante (del inglés: *Travelling salesman*)
- Enrutamiento de vehículos agrupados (del inglés: *Vehicle routing grouped*)

En cuanto a las estrategias heurísticas de bajo nivel que se podían producir en HyFlex, se pueden clasificar en cuatro categorías [9] [53]:

- Heurística de perturbación: La estrategia realiza pequeños cambios a la solución, usualmente siguiendo esquemas como intercambio elementos, modificación, agregar o eliminar componentes en la solución.
- Heurísticas deconstrucción y reconstrucción: en este enfoque, se intenta eliminar ciertas características de la solución, y después volver a generar los elementos eliminados usando heurísticas de construcción de bajo nivel.
- Heurísticas de búsqueda local o ascenso de la colina: en este tipo de técnicas, como su nombre lo indican, se buscan soluciones mejores a la actual.
- Heurísticas de mezcla: en esta estrategia se seleccionan al menos dos soluciones distintas y se busca producir una nueva solución a partir de la elección de candidatos.

Para cuantificar a las hiper-heurísticas se realiza el siguiente procedimiento: primero se hace una ponderación de los resultados en cada una de las instancias del problema que se trabaje. Después se hace un ordenamiento de acuerdo a su

desempeño, y se elige a las ocho estrategias con el mejor rendimiento para el problema de dominio.

A cada una de esas heurísticas se le asigna un rango, de manera que el rango más alto es la que obtuvo un mejor desempeño y el rango más bajo a la octava hiper-heurística, finalmente las hiper-heurísticas por debajo de las ocho primeras no tienen puntuación.[12] [53]

Lo anterior es realizado para cada uno de los seis problemas de dominio. Para finalizar, se realiza una suma de los rangos obtenidos como indicador del rendimiento total de la hiper-heurística sobre el desafío heurístico de búsqueda de domino cruzado (el método de evaluación anteriormente descrito se le denomina *prueba de suma de rango de Wilcoxon*) [9] [12].

Para finalizar otro software similar Hyflex es Hyperion el cual fue propuesto en 2011, Hyperio además de proporcionar soporte de desarrollo hiper-heurísticas, también ofrece herramientas para la implementación de heurística, metaheurística y algunos tipos de hibridación [12].

3.5 Aplicaciones

3.5.1 El problema de enrutamiento de vehículos

El problema de enrutamiento de vehículos (por sus siglas en inglés, VRP) es uno de los problemas de optimización combinatoria más investigado, debido a dos factores primordiales: la complejidad del problema lo vuelve ideal como campo de estudios para las ciencias computacionales, y por las aplicaciones que tienen en el mundo real como lo puede ser en cuestiones de logística [9] [12].

El planteamiento básico del problema VRP, consiste en generar un conjunto de rutas cercanas a un depósito. Donde cada vehículo tiene una capacidad asociada [15]. En cada una de las trayectorias, el vehículo debe de cumplir con una demanda de artículos solicitada por los consumidores. El objetivo es minimizar la distancia total y satisfaciendo la demanda de los consumidores. Cabe señalar que existen variantes a partir de la descripción dada, pero para propósitos de ilustración y aplicación del algoritmo, basta con el ejemplo dado [10] [11].

3.5.2 Problemas de horarios para personal

Los problemas de horarios para personal surgen en varios escenarios del mundo real, en donde se tenga una diversidad de recursos y se tenga la necesidad de una adecuada administración de cada elemento [15]. Algunos ejemplos pueden ser los horarios del personal de un supermercado, los horarios del personal de un centro de llamadas, los horarios para las enfermeras, por mencionar algunos [10] [11].

3.5.3 Problemas de horarios para instituciones educativas

Problemas de horarios para instituciones educativas en cualquiera de sus variantes UCTP (por sus siglas en inglés: el problema de horarios y cursos universitarios) [8], ETP (por sus siglas en inglés, el problema de horarios de exámenes) [54] [55] o CBCT (por sus siglas en inglés, el problema de horarios y cursos basados en currículum) [38] por mencionar algunos, han sido ampliamente utilizadas las hiper-heurísticas.

En general los problemas de horarios para instituciones educativas, consisten en asignar un conjunto de recursos (cursos, profesores, alumnos, etc.) a

un horario, el cual usualmente está compuesto de salones y horas hábiles como principales características [8]. Cabe destacar que son los propios recursos y la escasez de los mismos, los que provocan conflictos unos con otros [8].

Capítulo 4

4. El problema de horarios y cursos basado en currículum

Cuando se presenta un problema que, dadas sus exigencias, requiere la administración de diversos elementos (recursos humanos, infraestructura, tiempo, etc.), se presenta el dilema de cómo realizarlo de manera eficiente. Una de las formas en las que se ha podido dar una solución, es por medio de la generación de horarios que nos permitan obtener una adecuada gestión de los recursos [17] [25].

Algunos de los escenarios donde se pueden presentar este tipo de problemas son: hospitales; los cuales tienen recursos que deben ser gestionados: enfermeras, habitaciones de pacientes, equipo médico, etc., con la intención de no saturar de trabajo a las enfermeras y doctores, a la par de hacer un uso adecuado de la infraestructura de la institución médica (*nurse rostering problem*) [56].

Otra situación puede ser para las compañías en donde deben de manejar los elementos como: artículos de venta, operarios, puntos de venta, inventario, etc. El proponer un horario de laboral para los empleados donde la finalidad es tener una adecuada carga de trabajo para cada obrero, se vuelve una necesidad para la empresa (*jobshop scheduling problem*) [10].

Un último ejemplo son las instituciones educativas, en las cuales se deben de administrar: salones, profesores, alumnos, cursos, eventos escolares, etc. La manera en las que las instituciones educativas han abordado el problema es por medio de la generación de horarios para profesores y alumnos [25].

Cabe destacar que particularmente para las instituciones educativas, la generación de horarios es un área de interés, debido a que durante cada nuevo periodo escolar deben de reorganizar una gran cantidad de recursos. Por lo que proponer un horario donde se contemplen cada uno de los elementos, no es una tarea trivial [25].

Por no mencionar que cada institución educativa tiene recursos y necesidades diferentes, por lo que conseguir una adecuada gestión de los recursos se vuelve una tarea vital para el organismo [17]. Por mencionar un ejemplo, si una institución ofrece una gran cantidad de cursos, puede caer en una situación financiera inadecuada, por el contrario, si hay una oferta pobre de cursos, puede ocurrir una deficiencia en el nivel de educación de los alumnos [25].

Es aquí donde el problema de horarios y cursos basado en currículum abre una oportunidad como un acercamiento a las dificultades que tienen las instituciones educativas (particularmente las universitarias) para resolver los problemas relacionados con horarios o calendarización. El esquema CBCT fue propuesto por la Competencia Internacional de Horarios en año 2007, y es hasta la fecha un problema que continúa siendo estudiado [17].

El planteamiento propuesto por CBCT no es el primer esquema para abordar los problemas de calendarización para instituciones educativas [28] [40], pero se diferencia de otros modelos debido a que su enfoque se encuentra hacia el lado de la institución, generando el horario con base en los cursos que deben ser ofertados en el ciclo escolar, a diferencia de otros acercamientos como lo podría ser el problema de UCTP (por sus siglas en inglés: el problema de horarios y cursos universitarios), el cual propone la elaboración de horarios con relación a los alumnos y sus cursos [57].

La idea esencial de CBCT, es el alojar un conjunto de lecturas de cursos (la lectura es el equivalente a una clase) en diferentes periodos de tiempo, mientras se respetan una serie de restricciones de carácter duras y blandas que deben ser cumplidas al momento de proponer un horario [58] [59].

Las restricciones duras son limitaciones de carácter obligatorio, por lo que todas deben ser cumplidas al momento de proponer un horario para que sea considerado como válido [60]. Un ejemplo de restricción dura que es básica en cada uno de los esquemas de la Competencia Internacional de Horarios, es el que no existan dos eventos ocurriendo en el mismo periodo y lugar en el calendario [61].

Las restricciones blandas (también llamadas suaves) son limitaciones de carácter opcional, por lo que no todas deben ser cumplidas al generar un horario [57]. Un ejemplo de restricción suave sería asignar eventos a determinadas habitaciones, que dadas sus propiedades son adecuadas para el calendario. Usualmente tanto restricciones blandas como duras se encuentran en conflicto, por lo que el reto es encontrar un horario en el que se tengan los mayores beneficios [16] [46].

Una de las características más importante del problema CBCT son las bases de datos. Para cada esquema propuesto por la ITC, la competencia proporciona una serie de instancias que provienen de diferentes universidades alrededor del mundo, con la determinación de que el problema refleje las necesidades de instituciones reales [61].

Una de las características más importante del problema CBCT son las bases de datos. Para cada esquema propuesto por la ITC, la competencia proporciona una serie de instancias que provienen de diferentes universidades alrededor del mundo, con la determinación de que el problema refleje las necesidades de instituciones reales. Además, la Competencia Internacional de Horarios libera

varias instancias con la intención de tener un amplio espectro del comportamiento del algoritmo al momento de abordar el problema [61].

Cabe destacar que el problema CBCT, es un problema de optimización de tipo NP-Duro [14][16] [34], por lo que el problema se vuelve un área de interés para las ciencias de la computación y sus algoritmos [20].

4.1 Historia

Para hablar de CBCT, se debe de comenzar hablando de la Competencia Internacional de Horarios. El cual es un organismo que busca el fomento, desarrollo e intercambio de propuestas para resolver los problemas de horarios, particularmente los relacionados con los horarios para instituciones educativas [61].

La primera competencia se realizó en el año 2002, donde se propuso el *problema de horarios y cursos universitarios* (referido como UCTP y del inglés: *University Course Timetabling Problem*), el primer acercamiento de la organización para abordar problemas de calendarización para instituciones educativas. El problema contó con un total de 20 instancias para resolver y tuvo una participación de diversos investigadores con variadas propuestas para abordar la tarea [62].

La segunda competencia fue realizada en el año 2007, donde la ITC propuso una serie de tres nuevos esquemas para tareas de calendarización. Cada uno con planteamientos diferentes, pero siempre guardo relación con los problemas de horarios para instituciones educativas [61].

Dentro de los nuevos esquemas propuestos, se encuentra *el problema de calendarización de exámenes* (del inglés: *Examination Timetabling Problem*), el cual tiene como propósito la generación de un horario para la aplicación de exámenes [26].

El otro esquema fue una extensión del propuesto en 2002, *el problema de horarios y cursos universitarios,* donde se renombró como *el problema de horarios y cursos posterior a la inscripción* (del inglés: *Post Enrolment Based Course Timetabling*), y como adición al original, se incorporaron una nueva serie de restricciones. Cabe resaltar que el modelo se enfoca a la generación del calendario con base al horario que tendrán los alumnos [13].

Finalmente, el último esquema propuesto fue el problema de horas y cursos basados en el currículum, donde se prioriza la generación de un calendario de acuerdo a las materias a ofertar y profesores disponibles en el periodo escolar [58].

La ITC realizó otra competencia en el 2011 donde se abordó las tareas de calendarización para instituciones académicas de grado media-superior (del inglés: *High-School Timetabling Problem*) y en el 2019, donde se retomó solucionar las tareas de horarios de las universidades [63] [64].

4.2 Descripción de el problema de horarios y cursos basados en currículum

De manera general, el problema de horarios y cursos basado en currículum consiste en generar un horario en el cual deben de estar asignados todas las lecturas de los cursos, en las horas disponibles del calendario y en los salones que sean adecuados. De tal manera que cada lectura tenga un salón y una hora definidos [57] [58].

Cabe resaltar que CBCT tiene una serie de restricciones que deben ser satisfechas al momento de proponer un horario. Las restricciones se pueden dividir en dos categorías: restricciones duras y restricciones blandas. Las primeras son de

carácter obligatorio y las segundas son de carácter opcional, pero se debe satisfacer el mayor número de las mismas [59] [60].

4.2.1 Variables

El problema de horarios y cursos basado en currículum, maneja diversas variables que representan recursos que una institución educativa debe manejar: profesores, cursos, salones, tiempo, etc., por lo que para una clara ilustración del problema, se hará una descripción de cada una [60].

La forma de dividir el tiempo puede ser representado con las siguientes variables: día, hora y periodo de tiempo. Los días representan la semana de trabajo de la institución educativa (típicamente 5 a 6 días), mientras que las horas son los periodos de actividad que tiene la institución para realizar las labores, finalmente los periodos se obtienen al multiplicar el número de horas por el número de días [17] [59].

Los cursos son las materias que oferta la institución educativa en cada nuevo periodo escolar. Cada curso consta de un número fijo de lecturas que deben ser programadas en distintos periodos de tiempo, además cada lectura es atendida por un número definido de estudiantes y es impartida por un profesor. Como última característica, cada curso tiene un número mínimo de días en los que debe de ser extendidas las clases en el calendario [17] [58].

Los salones son las estancias donde se ofrecen las clases (laboratorios, auditorios, sala de conferencias, etc., no son consideradas), de acuerdo al planteamiento del problema, cualquier cuarto puede albergar a cualquier lectura. Cada habitación cuenta con diferente número de sillas, lo que constituye la propiedad principal de las aulas [58] [59].

Los profesores son las personas que ofrecen los cursos que propone la institución educativa. Cada maestro puede tener uno o más cursos asociados. Como ultima propiedad, existen profesores que no estarán disponibles a determinadas horas dentro del calendario [57] [60].

La última variable del problema es la *currícula*, la cual se define como un grupo de cursos que poseen la propiedad de que, al tomar cualquier par del conjunto, tienen un grupo de estudiantes que cursas ambas materias [17] [57].

4.2.2 Restricciones

El problema CBCT posee una serie de limitaciones que se deben considerar al momento de proponer un horario. Las restricciones se pueden dividir en dos categorías: restricciones duras y restricciones blandas (también llamadas restricciones suaves) [59].

Una restricción dura se define como un requisito que debe ser cumplido de manera obligatoria. Las restricciones duras son las que hacen que una solución sea válida sí cumple con todas las limitantes, y en caso contrario, no es viable y carece de valor para el problema. Particularmente para CBCT, se tiene las siguientes restricciones duras [17]:

1) Todas las lecturas pertenecientes a los cursos deben de ser asignadas en el calendario.
2) Dos lecturas no pueden ocurrir en el mismo periodo de tiempo y salón.
3) Lecturas pertenecientes a una currícula o impartidas por el mismo profesor, deben de ser asignadas en periodos diferentes.
4) Si un profesor no está disponible para impartir una lectura de un curso en un periodo de tiempo específico, entonces, no es posible asignar la lectura a esa hora.

Como se puede observar, la primera restricción dura se busca que no quede ninguna lectura fuera del calendario. En cuanto a la segunda restricción, representa la indicación de que no existan empalmes en los salones; es decir, que los periodos de tiempo y aula difieran de lectura en lectura [58].

En cuanto a la tercera restricción dura, se debe partir de una definición dada por la ITC2007: “una currícula es un conjunto de cursos donde; cualquier par de cursos en el grupo tienen estudiantes en común”, por lo que si se asigna dos cursos pertenecientes a una currícula en el mismo periodo de tiempo (aunque sea en salones distintos) sería el equivalente a solicitar a los alumnos estar en dos habitaciones distintas a la misma hora. Lo mismo aplicaría si el profesor es el responsable de impartir las materias contenidas en la currícula [59].

Finalmente, la última restricción dura nos indica que sí un profesor no se encuentra en disposición para ofrecer una lectura en un determinado periodo, no es posible asignar esa clase a dicha hora, pero puede ser asignada la lectura a otro periodo de tiempo con lo que se respetaría la restricción [57].

Una restricción blanda (también llamadas suaves) es un requisito de carácter opcional, por lo que no es necesario cumplir con todas al momento de proponer una solución. Sin embargo, cada restricción blanda no satisfecha genera un impacto sobre el valor del calendario al ser evaluado por la función objetivo. Por lo que cumplir con un mayor número de restricciones blandas ofrece al calendario un mejor valor y por tanto un candidato superior a ser empleado [17].

1) Para cada lectura; el salón asignado debe contar con número de asientos igual o mayor a los estudiantes.
2) Las lecturas de un curso deben ser extendidas en un mínimo de días.
3) Todos los cursos deben de estar en un sólo salón.

4) Lecturas pertenecientes a una currícula, deben de estar en periodos consecutivos en el mismo día.

La primera limitación nos indica que el salón donde se asigne la lectura debe de tener la capacidad suficiente para albergar a todos los estudiantes. En este caso particular, si la clase es asignada en un salón con una menor cantidad de asientos a la solicitada, el costo por no cumplir la restricción se calcularía como el número de sillas necesarias para alojar a los alumnos [60].

La segunda restricción blanda nos solicita que todas las lecturas deben ser extendidas en el calendario, esto viene a significar que sí un curso consta de dos lecturas, las lecturas deben ser asignadas en dos días distintos (Lunes y Martes por ejemplo), en caso de que tuviera tres lecturas, entonces se debe alojar en tres días diferentes (e. g. Lunes, Miércoles y Viernes). Usualmente por cada día repetido, se tiene un coste de más uno [58].

La restricción suave número tres nos pide que todos los cursos deben de estar asignados en el mismo salón (sin importar si son en periodos diferentes y en días distintos), en este caso, por cada salón distinto se añade una penalización de más uno [57].

Finalmente, la última limitación nos indica que los cursos pertenecientes a una currícula, idealmente deben estar en periodos consecutivos en el mismo día, con la intención de que los alumnos al salir de una clase puedan pasar a su siguiente hora de estudio (comprimiendo las horas que un estudiante debe de estar en la institución educativa). Aquí ocurre la penalización en caso de que una lectura se encuentre aislada [59].

4.2.3 Evaluación

Dado que es posible obtener más de un calendario que intente resolver el problema CBCT, se ve la necesidad de poder calificar la utilidad que tiene cada horario propuesto. Por lo anterior descrito la ITC2007 proporciona una serie de pautas que dictan cómo se debe realizar la valoración de una solución [17].

La calidad de una solución puede ser evaluada en términos de violaciones de restricciones duras y blandas. Particularmente para CBCT, todas las restricciones duras deben ser satisfechas, de otro modo la solución no tiene utilidad para el problema [14].

Dependiendo de la representación de la solución, es posible que la forma de realizar la evaluación cambie, sin embargo, en esencia lo que se realiza, es la ponderación de restricciones duras y restricciones blandas que no son satisfechas en el horario propuesto (Fórmula 1) [8].

$$CostoTotal = [\sum_{i=1}^{RD} P_D \cdot CostoD_i] + [\sum_{j=1}^{RB} CostoB_j]$$

Ecuación 1. Evaluación de una solución.

Donde:

RD: Son todas las restricciones duras a revisar en el calendario.
RB: Son todas las restricciones blandas a revisar en el calendario.
P_D: Es la penalización por romper una restricción dura.
$CostoD_i$: Es el costo asociado por romper una restricción dura.
$CostoB_j$: Es el costo asociado por romper una restricción blanda.

Cabe señalar que la Ecuación 1, debe ser aplicada a una solución propuesta para solucionar el problema de horarios propuesto por la ITC2007.

4.3 Propuestas para abordar el problema de horarios y cursos basados en currículum

El primer intento para abordar el problema CBCT, fueron los algoritmos propuestos en la primera competencia del año 2007. El ganador fue Tomás Müller, quien utilizó una estrategia híbrida donde se implementaban diferentes algoritmos que trabajaban en conjunto de los que destacan: Gran Diluvio, Recocido Simulado y Ascenso de la Colina [17].

Sin embargo, desde la publicación de los resultados iniciales de la competencia han surgido diferentes propuestas para abordar el problema CBCT, con resultados variables, en la siguiente sección se hará mención de algunos de los algoritmos más recientes para hacer frente a la tarea.

4.3.1 Recocido simulado

La metaheurística llamada recocido simulado fue propuesta por Kirkpatrick [44], está inspirada en un método empleado en la metalurgia denominado recocido. La técnica del recocido de metales consiste en elevar la temperatura del elemento y después de manera lenta, bajar la temperatura propiciando un arreglo de moléculas equilibrado y generando un estado de energía mínima [2].

De manera general, la metaheurística emula el proceso del recocido, al comienzo del algoritmo se inicia con una solución a una temperatura muy alta, lo cual, produce candidatos en los que se generan grandes cambios en su “estructura”. Conforme la temperatura desciende, los cambios son cada vez más finos hasta llegar a la temperatura de enfriado, simulando la configuración de baja energía [30] [35].

El funcionamiento del algoritmo inicia generando una solución inicial S (ya sea aleatoria o construida siguiendo una estrategia) e inicializando un parámetro de temperatura T. Después, en cada iteración, una solución S' es seleccionada aleatoriamente del vecindario $N(S)$de la actual solución S [2] [45].

La solución S' puede ser aceptada dependiendo de los parámetros de temperatura T y de los valores de la función objetivo de Sy S'denotado como $f(S)$ y $f(S')$respectivamente. Si se cumple $f(S') \leq f(S)$ entonces la solución es aceptada y reemplaza a S como solución actual [2] [45] [65].

Sin embargo, si sólo se aceptaran soluciones que cumplan la condición de ser mejor o igual a la original, podría derivar en una convergencia rápida, evitando una adecuada exploración del espacio solución del problema. Por lo que el algoritmo contempla este escenario e implementa un método que le permite explorar las demás regiones, el nombre del mecanismo es el criterio de Metrópolis [30] [35].

Para evitar una convergencia prematura hacia un óptimo local, las soluciones con un desempeño menor al actual también pueden ser admitidas, utilizando el criterio de Metrópolis. De acuerdo al criterio de Metrópolis, se puede aceptar una solución con una probabilidad de $P = e^{(-\Delta/T)}$ donde Δ es el cambio de calidad, mientras que T es el parámetro de temperatura, el cual es empleado para regular la probabilidad de aprobar soluciones con calidades menores [2] [45] [65].

Cabe destacar el manejo del parámetro de temperatura T durante el proceso de búsqueda; al comienzo de la inspección, la temperatura es alta lo que indica una mayor probabilidad de aceptar candidatos que degraden la solución actual. Esto puede ser visto como exploración del espacio de búsqueda [1] [2].

Con forme va avanzando el proceso de optimización, la temperatura descenderá de manera controlada. Cuando la temperatura comienza descender,

indica que las soluciones peores a la original son cada vez menos probables de ser aceptadas. Esto puede ser visto como inspeccionar con más detenimiento en áreas cada vez más prometedoras [1] [2].

La velocidad a la que desciende la temperatura para modificar la posibilidad de admitir soluciones deficientes se le denomina “esquema de enfriamiento”. Una de las maneras de más comunes de reducir la temperatura es aplicar el esquema de enfriamiento geométrico: $T_{i+1} = T_i * \alpha$ donde α puede ser obtenida de manera empírica de acuerdo al problema particular que se aborda [2] [45].

Con todo lo explicado anteriormente, se puede mostrar un esquema general del funcionamiento del recocido simulado [2]:

RecocidoSimulado:
s = generar_candidato()
T = Inicializa_temperatura()
MIENTRAS condición_de_paro():
REPETIR:
Selecciona de manera aleatoria $s' \in N(s)$
SI $f(s') \leq f(s)$ ENTONCES:
$s \leftarrow s'$
EN OTRO CASO: // criterio metrópolis
$s \leftarrow s'$ con una probabilidad de $p(T, f(s'), f(s))$
Enfriar (T)
HASTA “equilibrio térmico” alcanzado
REGRESAR candidato

Como se puede observar, es posible una pérdida de conocimiento durante el proceso de búsqueda, inclusive en caso de que la mejor solución s^* sea encontrada, puede ser reemplazada por otro candidato. Para este tipo de situación, es recomendable almacenar la mejor solución en el proceso de optimización [2] [45].

4.3.2 Gran diluvio

El algoritmo de gran diluvio es una estrategia de búsqueda propuesta por el investigador Duek, el cual funciona utilizando tres componentes principales: nivel de agua, ratio de decaimiento y solución esperada [39].

La técnica funciona de manera similar a recocido simulado, aceptando soluciones mejores a la actual y con la posibilidad de permitir soluciones peores. A diferencia de recocido simulado que se inspira de la técnica metalúrgica del recocido de metales, gran diluvio hace una analogía del ascenso y descenso del nivel de agua durante una lluvia prolongada [1][67].

Para el funcionamiento de gran diluvio básico se requiere de un mecanismo llamado ratio de decaimiento y el nivel del agua inicial. El nivel del agua es el elemento en el que se basa el algoritmo para aceptar una solución [39].

El valor del nivel del agua es inicialmente asignado en base al costo de una solución y eventualmente será reducido durante el proceso de búsqueda. Para lo cual, antes de la ejecución de gran diluvio, una estimación de calidad (la aptitud desea en la solución final) necesita ser fijado por el usuario. El algoritmo entonces explora las regiones que tienen costos similares a la valoración de la calidad búsqueda [66][67].

Se debe destacar que no hay garantía de que la solución resultante tenga el valor de costo planteado inicialmente. De hecho, puede haber escenarios donde la solución final tenga una calidad peor a la deseada o incluso una calidad superior a la estimada [66].

La razón de partir de una consideración de valor inicial es debido a que es empleado en el mecanismo de ratio de decaimiento. La ratio de decaimiento es una función que consta de tres parámetros: el valor de la solución actual, el costo

estimada (la aptitud deseada al final de la búsqueda) y el número de iteraciones [34][67].

A continuación, se muestra el pseudocódigo de gran diluvio propuesto por Dueck [39]:

Gran Diluvio:
Escoger una configuración inicial
Escoger una "velocidad de lluvia" ARRIBA > 0
Escoger el inicio de NIVEL_AGUA > 0
Opt: Elegir una nueva configuración la cual es una perturbación
estocástica pequeña de la vieja configuración
computa E = calidad (nueva configuración)
SI E > NIVEL_AGUA:
vieja configuración = nueva configuración
NIVEL_AGUA = NIVEL_AGUA + ARRIBA
SI mucho tiempo sin incrementos en calidad O muchas iteraciones:
detener
IR Opt

Gran diluvio es una técnica que no ha sido ampliamente adoptada como el recocido simulado, sin embargo, en los algoritmos que ha sido empleado ha generado resultados competitivos, aunque sea solo como motor de búsqueda global para un algoritmo híbrido [34].

4.3.3 Algoritmos genéticos

La computación evolutiva, es una rama de la inteligencia artificial que agrupa mecanismos y técnicas basadas en la evolución natural y la genética; tomando

como una de sus bases teóricas la "*Teoría de la Evolución de las Especies*" de Charles Darwin, los estudios realizados por Gregor Mendel en la genética y fenómenos en la naturaleza, siendo la técnica más representativa del rubro el *algoritmo genético* propuesto por John Holland [42].

Reciben el nombre de algoritmo genético debido a la forma en que trabajan. En su diseño tienen mecanismos internos denominados *operadores genéticos* [6], los cuales intentan emular el proceso de selección natural, la modificación genética, la evolución de las especies entre otros, con el fin de mejorar la población con el pasar de las generaciones [38].

A diferencia de otras heurísticas como *la escalada de la colina* donde una solución inicial es mejorada en cada iteración [30] [35], los algoritmos genéticos producen un conjunto de posibles candidatos en cada iteración que realizan [43].

La capacidad de generar soluciones distintas desde el comienzo de la ejecución, permite al algoritmo el no estancarse de manera prematura en un óptimo local, además de generar una exploración más amplia del campo solución, debido a que idealmente cada solución se encuentra en regiones diferentes [43].

La técnica trabaja con una población de soluciones, las cuales tienen una escala de *adaptación* para resolver el problema [6]. Siguiendo con la analogía de la naturaleza, los *cromosomas* de los individuos van siendo modificados cada vez que transcurre una *generación* dentro del algoritmo genético [43].

Como ocurre en la naturaleza, los individuos con mejor grado de adaptación tienen una mayor probabilidad de sobrevivir y procrear a la siguiente generación, en los algoritmos genéticos se imita este proceso en los denominados operadores genéticos [38] [43].

Los operadores genéticos tienen por objetivo, la mezcla entre los individuos para la generación de nuevos candidatos que puedan mejorar a la población actual, sin embargo y como ocurre en la naturaleza, la aplicación de dichos operadores genéticos no garantiza un incremento en el grado de adaptación con respecto a las soluciones originales [6] [38].

De la misma manera que la naturaleza está en constante cambio, explorando nuevas series de adaptaciones para los individuos, los mecanismos internos del algoritmo genético están continuamente buscando áreas prometedoras y eventualmente irán desapareciendo las regiones que no son tan favorables para la resolución del problema [6] [43].

Hay muchas formas de diseñar un algoritmo genético, gracias a los avances y el desarrollo que ha tenido está técnica, los investigadores tienen un conjunto diverso de operadores genéticos y mecanismos para implementar en su algoritmo [43], aunque un esquema esencial de la estrategia contempla los siguientes métodos y estructura en general [6]:

```
AlgoritmoGénetico:
    población = generar_población_inicial()
    evaluación(población)
    MIENTRAS: condicion_de_paro (población); HACER
        candidatos = selección(población)
        nuevos_canditados = cruzamiento(candidatos)
        mutación(nuevos_candidatos)
        población = nuevos_candidatos
        evaluación(población)
    REGRESAR mejor_individuo(población)
```

De manera breve y resumida, se explicará en que consiste cada mecanismo y operador en el pseudocódigo del algoritmo genético [6][41][42]:

Generar población inicial: es un mecanismo el cual construye una serie de individuos (usualmente de manera aleatoria) con el objetivo de dotar al algoritmo

de una variedad de soluciones para poder explorar un mayor espectro del espacio de búsqueda [6][41].

Selección: tiene el propósito de escoger a los individuos que dado su grado de adaptación, pueden tener la posibilidad de transmitir sus genes, dando mayor número de posibilidades a los individuos que son más aptos [6][42].

Cruzamiento: es un procedimiento que mezcla los genes de los individuos, para producir nuevas soluciones [41][42].

Mutación: método que consiste en generar una ligera variación de forma aleatoria en los genes del individuo con el objetivo otorgar una nueva característica a la solución [6][41].

Como se puede observar, puede existir la pérdida de conocimiento durante la ejecución, por lo que suele integrarse un mecanismo que almacene el mejor candidato encontrado en todo el proceso de búsqueda [41][42].

Han sido aplicado de manera exitosa los algoritmos genéticos para el problema CBCT generando resultados aceptables, al poseer una gran diversidad de operadores genéticos [43] les permite tener un abanico de herramientas para resolver el problema [8] [38].

4.3.4 Hiper-heurísticas

Las hiper-heurísticas buscan ser una técnica de mayor grado de generalización que las metaheurísticas. En los algoritmos hiper-heurísticos se plantea obtener resultados consistentes en diferentes instancias de diversos problemas, en lugar de obtener resultados excelentes en pocas instancias de un solo problema de dominio [9] [10].

En el esquema básico de las hiper-heurísticas existen dos componentes principales: el nivel alto y el nivel bajo. En el nivel inferior se encuentra las heurísticas de bajo nivel, las cuales trabajan directamente con el problema. Mientras que en el nivel superior se encuentra una estrategia que guía el proceso de búsqueda. Por último, existe el componente denominado barrera de dominio, el cual tiene el objetivo de separar ambos niveles. así como ser el medio de intercambio de información entre los mismos [11][12].

Una de las características más sobresalientes de las hiper-heurística es su capacidad de realizar una búsqueda sobre el espacio heurístico en lugar del espacio solución típico. Esto es posible debido a que tienen a su disposición un conjunto de heurísticas de bajo nivel, las cuales trabajan directamente con el problema, por lo que es responsabilidad del nivel superior elegir que heurísticas de nivel inferior tienen un mejor desempeño ante el problema que están resolviendo [9] [13].

Las hiper-heurísticas han tenido éxito abordando diferentes tipos problemas: horarios para personal de compañías, enrutamiento de vehículos, y por su puesto problemas de horarios para instituciones educativas de diversos niveles [10].

Particularmente para los problemas de horarios de instituciones educativas, han sido aplicadas en el problema de horarios y cursos universitarios (*university course timetabling problem)* [47], el problema de horarios de exámenes (*examination timetabling problem)* [26] y por supuesto en el problema de horarios y cursos basados en currículum (*curriculum based timetabling problem)* [14].

4.3.5 Algoritmos híbridos

En los problemas de optimización, cuando una estrategia no obtiene los resultados esperados, una opción viable para solventar las deficiencias de un acercamiento inicial, es la hibridación con otro algoritmo para potenciar las ventajas y reducir las carencias. Por tanto, el objetivo principal al integrar dos técnicas, es obtener mejores resultados trabajando en conjunto, a los que se podrían generar con los algoritmos por separado [27] [28].

Es ideal cubrir las deficiencias de un esquema con las fortalezas de otra estrategia y viceversa. La tarea no es trivial de lograr, de hecho, se pueden producir algoritmos donde el alto grado de complejidad vuelve difícil el mantenimiento del sistema y en el peor de los casos, no existen mejoras sustanciales al integrar las estrategias [27] [68].

Para el problema CBCT, parece ser adecuado emplear técnicas híbridas para obtener resultados competitivos. Generalmente empleado una estrategia de búsqueda global y una de búsqueda local, algunos autores han optado por utilizar incluso tres técnicas mezcladas con la finalidad de obtener mejores resultados a los que se podrían obtener por separado [69] [70] [71].

Capítulo 5

5. Algoritmo propuesto

Un algoritmo de fuerza bruta podría obtener el calendario con el menor costo, para cada una de las instancias del problema CBCT. Sin embargo, el estar evaluando cada uno de los posibles horarios de manera exhaustiva, tiene un costo de tiempo que no es posible disponer. De acuerdo con varios autores el problema CBCT es un problema NP-Duro [14] [16], por lo que se vuelve un problema de interés para las ciencias de la computación.

Por lo descrito en el párrafo anterior, se ha seleccionado a la hiper-heurística como estrategia para abordar el problema CBCT, por sus propiedades de flexibilidad y adaptabilidad a las diferentes instancias que presenta el problema [9]. Cabe destacar que las heurísticas de bajo nivel que posee el algoritmo, le permiten tener acceso a diversas herramientas para atacar al problema.

5.1 Representación de la solución

Para el funcionamiento de la hiper-heurística, es necesario que la representación de la solución sea lo más simple posible para que los tiempos de

ejecución del algoritmo sean aceptables, por lo que el calendario es representado como una matriz de MxN, donde M son los salones que disponemos para alojar los cursos, N son las horas disponibles en toda la semana. Ilustración 1.

	LUN			MAR		
	Hora 1	Hora 2	Hora 3	Hora 1	Hora 2	Hora 3
Salón 1						
Salón 2						
Salón 3						
Salón 4						
Salón 5						

Ilustración 1. Representación de la solución.

Los valores dentro de cada celda de la matriz son valores numéricos que representan las lecturas pertenecientes a cada curso. El rango de números a manejar se obtiene al serializar cada de las lecturas de cada curso, después se puede realizar un mapeo de los números con el objetivo de conocer cuales números representan las lecturas de los cursos.

Explicado con un ejemplo: se tienen dos cursos; el curso uno con doce lecturas y el curso dos con seis lecturas, entonces las lecturas del curso uno estará en el rango del cero al once, mientras que las lecturas del curso dos se encuentran en los números que van del doce al diecisiete. Imagen 2.

	LUN			MAR		
	Hora 1	Hora 2	Hora 3	Hora 1	Hora 2	Hora 3
Salón 1	0		9		8	
Salón 2		1				6
Salón 3	11		5	2		12
Salón 4	7		13		10	
Salón 5		4			3	

Ilustración 2. Inserción de cursos.

5.2 Generación de una solución inicial

Debido a que el algoritmo trabaja con una solución previamente generada, es menester mencionar cómo se construye la solución inicial. Existen dos corrientes principales al momento de proponer un calendario inicial: una donde la generación es completamente aleatoria y la segunda donde existe una estrategia para la construcción del candidato [9].

El primer enfoque ofrece una búsqueda más completa por el espacio de búsqueda, debido a que, en cada nueva ejecución, en principio, el algoritmo arrancara en una región distinta. Sin embargo, se tiene la desventaja de que puede tomar más tiempo llevar una solución de una región infactible a una región factible.

Mientras que la segunda estrategia tiene la intención de ayudar al algoritmo en su proceso de búsqueda, aunque eso represente un mayor esfuerzo al momento de iniciar el proceso de optimización. En el presente trabajo, se ha seleccionado la segunda estrategia.

Gracias a la representación de la solución en forma de matriz, algunas de las restricciones pueden ser garantizadas durante el proceso de optimización. Específicamente la restricción dura uno la cual nos indica que todas las lecturas deben de estar presentes en el calendario, y la restricción dura dos la cual señala que no es posible que dos lecturas ocupen el mismo periodo de tiempo y salón.

Lo anterior descrito es posible debido a que cada celda, sólo puede tener un número asociado, adicionalmente, se conoce de antemano todas las lecturas que deben ser colocadas, por lo que se garantiza que cada una de las lecturas tenga un lugar en el calendario.

La restricción dura número cuatro, puede ser manejada de la siguiente forma; antes de colocar una lectura de un curso, se debe revisar que el curso no tenga ninguna restricción de hora, en caso de no tener, el curso puede ser alojado en cualquier hora, en caso contrario, se evitará ubicar a la lectura en la hora no válida.

Por último, para finalizar con la construcción de la solución inicial, el orden de inserción de los elementos en el calendario se realiza de la siguiente forma; primero se ordenan los cursos de mayor a menor tomando como criterio el número de estudiantes pertenecientes al curso. Lo anterior se efectúa con la intención de priorizar los cursos con mayor cantidad de alumnos a las aulas que tengan la capacidad suficiente de sillas.

Puede existir la posibilidad de que aún haya cursos sin poder alojar siguiendo las reglas antes descritas (e.g. quedan lecturas sin alojar, pero se han acabado los salones con la capacidad de sillas necesarias), para esos casos, las lecturas restantes por asignar se colocarán de manera aleatoria, para respetar la restricción dura uno. Lo anterior es posible debido a que no existe limitación para almacenar un curso en cualquier salón.

De manera resumida, la generación de una solución inicial puede ser visualizada de la siguiente forma:

1) Realizar el rango de las lecturas de cada curso.
2) Colocar todas las lecturas en diferentes casillas del calendario de la siguiente forma.
3) Ordenar las lecturas de mayor a menor tomando como criterio el número de estudiantes.
4) Seleccionar en forma ordena los elementos de esa lista.
5) Revisar que la lectura no tenga restricción dura cuatro.

6) Insertar el curso en cualquier salón con la capacidad suficiente de asientos sin importar la hora.
7) En caso de no ser posible, insertar de manera aleatoria la lectura.
8) Repetir hasta no tener más lecturas.

El proceso anterior descrito no garantiza que todas las restricciones duras sean satisfechas (puesto que aún no se ha realizado ningún tipo de esfuerzo adicional para abordar la restricción dura número tres), sin embargo, mejora considerablemente el valor inicial del calendario.

5.3 Función de evaluación

Es esencial para posteriores pasos del algoritmo propuesto, un mecanismo que califique cuáles horarios presentan una mejora o degradación con respecto a la solución original. Por lo que el método para evaluar al horario será en principio, el mismo utilizado por ITC [17] (Fórmula 1), con la diferencia de que el costo asociado por romper una restricción dura es 300 veces mayor que al violar una restricción blanda (cuyo valor es de uno).

5.4 Nivel alto de la hiper-heurística

El nivel alto en una hiper-heurística tiene la responsabilidad de guiar el proceso de búsqueda en el espacio heurístico. Por lo que se ha seleccionado al recocido simulado como el nivel superior, principalmente por su capacidad de salir de óptimos locales gracias al mecanismo del criterio de Metrópolis [30].

Además de tener la responsabilidad de escapar de óptimos locales, el recocido simulado maneja dos variables más para cada heurística de bajo nivel; el número de veces seleccionada y el número de veces aceptada. La primera

almacena las veces que fue elegida, y después de su ejecución, en caso que de que haya sido aceptada su modificación a la solución, la segunda variable aumenta en uno, de otro modo no hay incremento.

Cabe señalar que la pérdida de conocimiento es un problema que se puede presentar en el recocido simulado, por la naturaleza de su funcionamiento para evadir óptimos locales. Teniendo en cuenta ese factor, se guarda la mejor solución encontrada en todo el proceso de optimización, siendo esa la que se devuelve al finalizar la ejecución.

5.5 Mecanismo de selección para las heurísticas de bajo nivel

En una hiper-heurística, los métodos de selección tienen el propósito de escoger a las heurísticas de bajo nivel, con base a una calificación que evalúa su desempeño. Con una calificación más alta, mayor posibilidad tendrá de ser elegida para trabajar en el proceso de búsqueda, con la finalidad de generar una aportación para resolver el problema.

Los procedimientos de selección, también deben de contemplar a las heurísticas de bajo nivel con un desempeño pobre, otorgándoles la posibilidad de ser elegidas en el futuro, aunque su rendimiento no sea el esperado.

La selección por ruleta, es un mecanismo clásico empleado por los algoritmos genéticos cuya forma de trabajar es la siguiente: a cada uno de los individuos que conforman la población, se les hace una evaluación empleando la función objetivo con el fin de obtener el grado de adaptabilidad [6].

Realizado lo anterior, se procede a sumas todos los grados de valor de aptitud, después se hace un ordenamiento de los individuos en base a su valor de

aptitud y se les asigna un segmento de acuerdo a su grado de adaptabilidad divido entre la suma de adaptabilidad total. Por lo que el primer individuo tendrá el rango que va de cero a su segmento, el siguiente individuo tendrá el rango que inicia con el segmento anterior y la suma de los segmentos anteriores más el propio, y así sucesivamente.

Realizado lo anterior, los candidatos con mayores grados de adaptabilidad recibirán un segmento más grande de la ruleta con respecto a la que recibirán los peores individuos. Cabe señalar que la suma de todos los segmentos debe de ser igual a la unidad.

Para seleccionar a un individuo, con la población previamente ordenada, se debe generar un número aleatorio en el intervalo [0, 1], y regresar el individuo situado en la posición correspondiente. La forma para obtener el individuo es: recorrer la población e ir acumulando sus segmentos de ruleta, hasta encontrar al candidato donde la suma sea igual o exceda al número aleatorio obtenido.

De manera similar a como funciona el mecanismo de ruleta para algoritmos genéticos [6], el algoritmo utiliza una estrategia que segmenta en rangos la posibilidad de que una heurística pueda participar en el proceso de optimización. La forma de realizarlo es la siguiente.

En principio, se desea que todas las heurísticas de bajo nivel participen a lo largo del proceso de optimización, lo anterior debido a que, en principio, aunque una heurística al comienzo de la búsqueda tenga un mal desempeño, esto no significa que después dicha heurística continué con ese mismo comportamiento.

Por lo que en todo momento las heurísticas deben tener un mínimo de posibilidad de ser seleccionadas, a dicha variable se nombrará *PorcentMínimo*, el cual tendrá un valor de 0.04. Debido a que cada heurística tiene un pequeño

porcentaje de ser aceptada, se debe considerar que no se tiene el cien por ciento de la ruleta. Para calcular el porcentaje restante, se emplea la Fórmula 1.

$$PorcentRestante = 1 - (PorcentMínimo * NumHeurísticas)$$

Fórmula 1.

Cada heurística tiene dos propiedades que pueden ser evaluadas durante el proceso de optimización: la cantidad de veces que fue seleccionada y el número de veces que su aportación fue de utilidad para resolver el problema. El *aporte* que cada heurística proporciona se definirá como el número de veces que fue aceptada sobre el número de veces seleccionada (Fórmula 2). En caso de que heurística de bajo nivel nunca haya sido seleccionada, entonces su aporte será cero. La suma de los aportes lo se denominará *AporteTotal*. Fórmula 3.

$$Aporte_h = Aceptado_j / Seleccionado_i$$

Fórmula 2.

$$AporteTotal = \sum_{k=1}^{Heurísticas_k} Aporte_h$$

Fórmula 3.

Se define a la contribución que cada heurística realiza como el resultado del aporte de la heurística sobre el aporte total de todas las heurísticas (Fórmula 4). Por último, para calcular el segmento que le corresponde a una heurística se puede obtener de acuerdo a lo descrito en la Fórmula 5.

$$Contribución_h = Aporte_h / AporteTotal$$

Fórmula 4.

$$Segmento_h = PorcentMínimo + (Contribución_h * PorcentRestante)$$

Fórmula 5.

Cabe señalar que el segmento es un valor acumulado, es decir: para la primera heurística su rango va de 0 a su segmento, el siguiente tendrá el rango que inicia con el segmento anterior y la suma de los segmentos anteriores más el propio, y así sucesivamente.

De manera resumida, la forma de calcular los segmentos de ruleta que tendrá cada heurística de bajo nivel se puede realizar de la siguiente manera:

1) Calcular el porcentaje restante (Formula 1).
2) Calcular el aporte realizado de cada heurística (Formula 2).
3) Calcular el aporte total (Formula 3).
4) Calcular la contribución de cada heurística (Formula 4).
5) Calcular el nuevo segmento de cada heurística (Formula 5).
6) Ajustar los nuevos segmentos de ruleta.

5.6 Heurísticas de bajo nivel

Las heurísticas de bajo nivel (llamadas así porque son las que modifican directamente a la solución) propuestas en el presente trabajo, tienen el propósito de realizar cambios en la respuesta con la intención de mejorar su resultado con respecto a la original. Sin embargo, también existe la posibilidad de que la perturbación realizada empeore la solución inicial.

La intención de las heurísticas de bajo nivel es conducir la búsqueda del algoritmo, por el espacio solución que existe. Cada heurística debe tener la intención de realizar una aportación al proceso de optimización, por lo que es recomendable utilizar varias heurísticas de bajo nivel para abordar diversos aspectos del problema. A continuación, se dará una descripción de cuales mecanismos fueron empleados.

- H1: Intercambiar una lectura en conflicto con otra lectura en conflicto en la misma fila.
- H2: Intercambiar una lectura en conflicto con otra lectura cualquiera en la misma fila.
- H3: Intercambiar una lectura en conflicto con una casilla vacía en la misma fila.
- H4: Intercambiar una lectura en conflicto con otra lectura en conflicto en la misma columna.
- H5: Intercambiar una lectura en conflicto con otra lectura cualquiera en la misma columna.
- H6: Intercambiar una lectura en conflicto con una casilla vacía en la misma columna.
- H7: Intercambiar una lectura en conflicto con otra lectura en conflicto en cualquier parte del calendario.
- H8: Intercambiar una lectura en conflicto con otra lectura cualquiera casilla vacía en cualquier parte del calendario.

Las tres primeras heurísticas de bajo nivel realizan cambios en la fila, las siguientes tres en las columnas y finalmente las últimas dos pueden realizar cambios en cualquier parte del calendario. Se han elegido estos mecanismos debido a la representación de la solución.

5.7 Condición de paro

Para el presente trabajo se ha seleccionado como condición de paro el establecer un margen de tiempo. Debido principalmente a que la ITC2007 establecido la utilización de 500 segundos de tiempo para cada instancia en su competencia [17].

Capítulo 6

6. Experimentación

6.1 Instancias

Para la parte de experimentación, se empleó la base de datos de la ITC2007 [17] para el problema CBCT, la cual es un referente para muchos investigadores del área. La base de datos consta de un total de 21 instancias, las cuales son distintas unas de otras en los valores de cada variable. La Tabla 1 muestra con mayor detalle la composición de la base de datos con algunos de los parámetros más importantes.

Instancia	Cursos	Lectura	Salón	Días	Horas	Periodo	Currícula	Horas Prohibidas
comp01	30	160	6	6	5	30	14	53
comp02	82	283	16	5	5	25	70	513
comp03	72	251	16	5	5	25	68	382
comp04	79	286	18	5	5	25	57	396
comp05	54	152	9	6	6	36	139	771
comp06	108	361	18	5	5	25	70	632
comp07	131	434	20	5	5	25	77	667
comp08	86	324	18	5	5	25	61	478
comp09	76	279	18	5	5	25	75	405
comp10	115	370	18	5	5	25	67	694
comp11	30	162	5	9	5	45	13	94
comp12	88	218	11	6	6	36	150	1368
comp13	82	308	19	5	5	25	66	468
comp14	85	275	17	5	5	25	60	486
comp15	72	251	16	5	5	25	68	382
comp16	108	366	20	5	5	25	71	518
comp17	99	339	17	5	5	25	70	548
comp18	47	138	9	6	6	36	52	594
comp19	74	277	16	5	5	25	66	475
comp20	121	390	19	5	5	25	78	691
comp21	94	327	18	5	5	25	78	463

Tabla 1. Descripción de la base de datos ITC2007.

Como se puede observar en la Tabla 1, las 21 instancias difieren unas de otras, pero se debe destacar que en cada una de ellas la ITC asegura que existe un horario en el que todas las restricciones duras son satisfechas, pero se desconoce si lo todas las restricciones blandas pueden ser cumplidas.

6.2 Parámetros del algoritmo propuesto

Es menester mencionar los valores usados para los parámetros utilizados en el recocido simulado y los indicadores relacionados con la selección de las heurísticas de bajo nivel. Los números para las variables se obtuvieron a base de experimentación aunado a la limitación de tiempo impuesta por la ITC en su competencia del año 2007 (500 segundos de ejecución).

Cabe señalar que, para determinar los valores de cada parámetro, se tomó cuatro instancias de prueba de la ITC2007 de manera aleatoria. Los ajustes de cada parámetro se realizaron de manera empírica priorizando obtener los mejores resultados posibles en las cinco instancias (en vez de obtener un muy buen resultado en sólo una o dos instancias) con la limitación de tiempo impuesta.

Parámetros para el recocido simulado

- Temperatura inicial: 2500
- Tiempo: 500 segundos
- Iteraciones: 1000
- Esquema de enfriamiento: geométrico
    - Alfa = 0.93

Heurística de bajo nivel

- Esquema de selección: basado en ruleta
- Probabilidad mínima de ser seleccionada: 0.04
- Número de heurísticas de bajo nivel: 6

6.3 Consideraciones adicionales

Con la finalidad de tener suficientes datos para evaluar el desempeño general del algoritmo para el problema y en particular en cada instancia de la base de datos, se realizaron un total de 10 experimentos para cada una de las instancias de la base de datos teniendo un total de 210 pruebas ejecutadas.

Como agregado a lo anterior, en la competencia del año 2007 la ITC señaló que cada algoritmo entregado sería ejecutado un total de 10 veces por cada instancia. Por lo que además de ser una medida para obtener datos estadísticos los 10 experimentos, también cumple el propósito de ser un número ideal de ejecuciones para compararnos con el estado del arte.

Con el propósito de tener un punto de referencia acerca del desempeño de la hiper-heurística con respecto a otras estrategias, se realizó una comparación con distintas técnicas encontradas en la literatura. El criterio para seleccionar a los algoritmos fue: ser propuestas menores a cinco años de su publicación, ser estrategias metaheurísticas e hiper-heurísticas y haber obtenido sus resultados con la base de datos de la ITC2007.

Cabe recordar que, para todas las pruebas, las soluciones iniciales fueron generadas empleando la técnica descrita en el funcionamiento del algoritmo. Lo anterior debido a que las soluciones generadas de manera aleatoria, obtenían resultados inferiores a los generados con la estrategia.

Por último, el programa fue codificado en el lenguaje de programación Python 3.6, además los experimentos fueron realizados en un equipo de cómputo

con las siguientes características: sistema operativo Linux 64 bits, un procesador Intel i5, con una capacidad de 8 GB en RAM.

Capítulo 7

7. Resultados

En las gráficas (Ilustración 3 e Ilustración 4) que se muestran a continuación se exponen los resultados obtenidos por la hiper-heurística propuesta (color verde punteado), además se hace una comparación con los valores obtenidos por otras estrategias encontradas en la literatura. En las gráficas, el eje *x* representa las instancias de la ITC2007 y el eje *y* representa el valor mínimo obtenido en los experimentos por los algoritmos.

Específicamente el programa fue comparado contra: un recocido simulado modificado [65], dos hiper-heurísticas [14] [53], una heurística que emplea grafos [37] para su funcionamiento y finalmente los mejores resultados obtenidos en la literatura revisada junto con la búsqueda de otros autores (color azul punteado) [72].

La primera estrategia (color azul) [65] es en esencia, un esquema clásico de recocido simulado con dos modificaciones importantes: el esquema de enfriamiento denominado *cutoff-based* y la condición de paro. El primer mecanismo desciende la temperatura en base al número de soluciones aceptadas, a mayor número de respuestas admitidas, más rápido el enfriamiento. El segundo método es una función la cual se compone de dos elementos principales: el número aproximado de soluciones a trabajar entre cada cambio de temperatura y la temperatura mínima a alcanzar, de tal manera que se logré llegar a la temperatura deseada al final del proceso y aprovechando al máximo el tiempo disponible [65].

La primera hiper-heurística (color naranja) [14] es de tipo generación-constructiva; en el nivel alto se localiza un algoritmo de programación genética, el cual propondrá heurísticas para abordar el problema, en cuanto al nivel bajo se encuentran: heurísticas para la selección de periodos del calendario, estrategias para el ordenamiento de lecturas de acuerdo a diferentes criterios y otras propiedades del problema. De modo que cada heurística generada es representada como una combinación jerárquica de características del problema y una heurística de selección de periodo [14].

La segunda hiper-heurística (color amarillo) [53] es de tipo híbrida, en su estructura se encuentra integrados varios elementos: una estrategia de búsqueda local (*Iterated Local Search*), elementos de la hiper-heurística agregar-eliminar (*add-delete HH*) [47] y un conjunto de diferentes heurísticas de perturbación de bajo nivel. Adicional a lo anterior, la estrategia también incorpora un método para la diversificación de las soluciones y un mecanismo para la generación de una heurística adaptativa para construir una solución inicial [53].

Finalmente, la última estrategia es una propuesta enfocada a crear una población de soluciones iniciales para CBCT empleando técnicas de grafos encontradas en la literatura (color verde) [37]. En la propuesta los autores se concentran en trabajar con tres tipos de estrategias: *largest degree* (lecturas que tienen conflictos con otras lecturas previamente asignadas), *weighted degree* (estudiantes en conflicto en el calendario) y *saturation degree* (lecturas con el menor número de periodos viables en el calendario) evaluando su desempeño por separado y en conjunto (máximo dos). De manera general, se ordenan los cursos de acuerdo a los criterios seleccionados y después se van colocando en el calendario hasta terminar con una solución construida [37].

En la ilustración 3, se muestran los valores mínimos obtenidos de la instancia comp01 a la comp10 por el algoritmo propuesto, así como los valores generados por otras estrategias, mientras que la ilustración 4, muestra las instancias restantes comp11 a la comp21 con las respectivas puntuaciones de cada una de las técnicas.

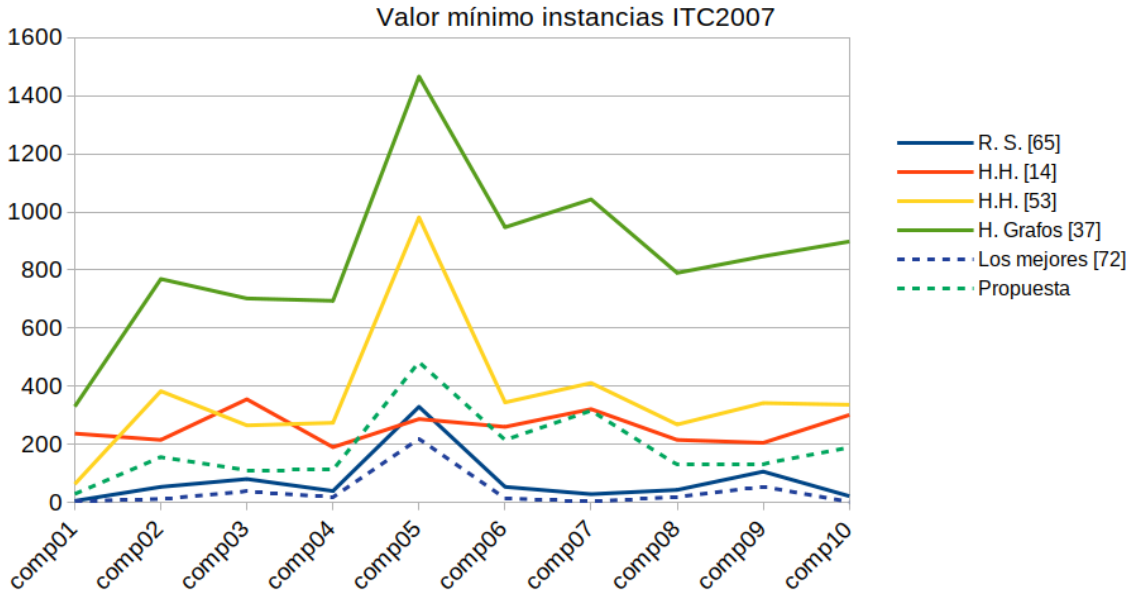

Ilustración 3. Resultados de las instancias comp01 a la comp10.

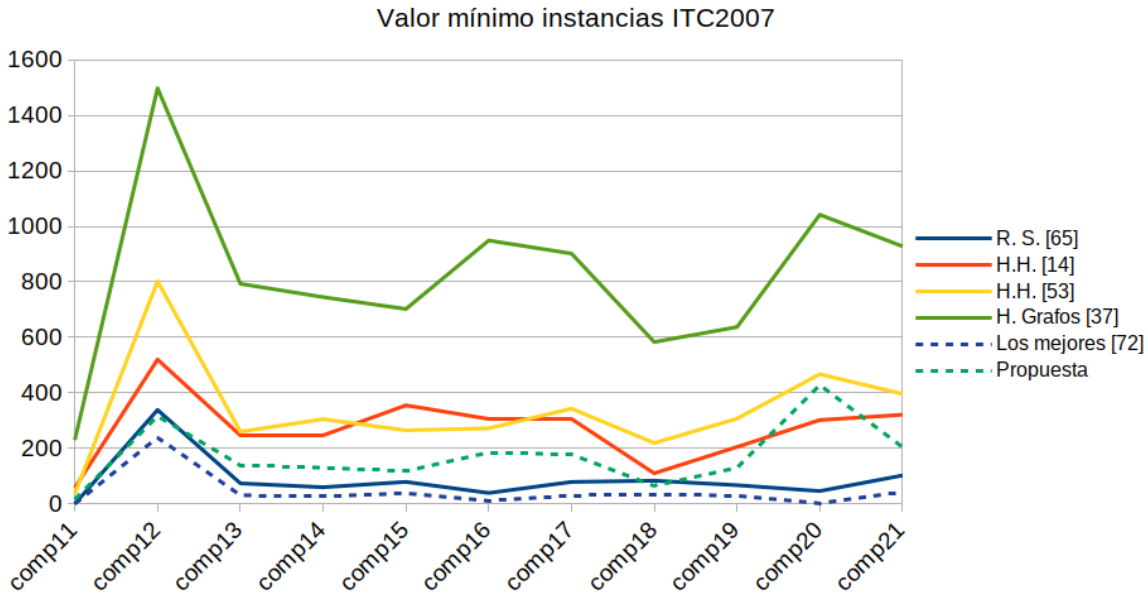

Ilustración 4. Resultados de las instancias comp11 a la comp21.

Como se puede observar en las ilustraciones 3 y 4, el desempeño general del algoritmo, tiende a ser mejor que el presentados por otras estrategias hiper-heurísticas. La combinación de los vecindarios junto con la selección de ruleta, permite obtener una mejora significativa.

Cabe mencionar que, en todas las instancias evaluadas, el algoritmo propuesto obtuvo mejores resultados en comparación con los valores obtenidos por la estrategia que emplea grafos. Sin embargo, los puntajes obtenidos por el recocido simulado modificado (salvo por la comp18), son superiores a los generados por la hiper-heurística propuesta.

En la Tabla 2 se muestran los valores puntales obtenidos por cada una de las estrategias. La primera columna corresponde a la instancia de la base de datos, en la columna posterior se describen los valores promedio del recocido simulado modificado, las siguientes dos muestran el mejor resultado encontrado por las hiper-heurísticas, después se muestran los valores reportados por la estrategia de grafos y las últimas dos muestran los valores mínimos obtenidos y la media de las pruebas respectivamente.

Cabe señalas que existen casillas en las que además del valor principal, se encuentra un número en paréntesis. El número en paréntesis indica las restricciones duras no satisfechas, mientras que el valor sin paréntesis, indica las restricciones blandas no cumplidas.

						Propuesta	
Instancia	R. S. [65]	H.H. [14]	H.H. [53]	H. Grafos [37]	Los mejores [72]	Valor Mínimo	Media
comp01	5.16	237	63	330	4	29	31.2
comp02	53.42	215	383	769	12	156	196.5
comp03	80.48	355	265	702	38	111	128.6
comp04	39.29	190	274	694	18	112	125.4
comp05	329.06	287	981	1466	219	483	(1.1) 555.7
comp06	53.35	260	344	947	14	215	243.2
comp07	28.45	321	411	1043	3	316	474.7
comp08	43.06	215	268	790	19	129	150
comp09	106.1	205	342	847	54	133	143.2
comp10	21.71	301	336	898	2	189	239.1
comp11	0	58	36	230	0	16	19.4
comp12	338.39	520	802	1498	239	316	374.6
comp13	73.65	246	260	793	31	139	154.7
comp14	59.71	248	305	745	27	130	141.2
comp15	79.1	355	265	702	38	118	141.7
comp16	39.19	306	272	949	11	185	256.3
comp17	78.84	308	343	902	30	178	201.6
comp18	83.29	110	219	583	34	65	69.5
comp19	67.13	205	307	637	29	130	148.5
comp20	45.94	302	467	1042	2	428	814.8
comp21	102.19	321	396	928	43	204	234.7

Tabla 2. Tabla de valores obtenidos por las técnicas.

Como se puede observar, los valores mínimos generados por el algoritmo, son en general menores a los que obtienen las otras dos hiper-heurísticas. Las únicas excepciones son las instanciaras comp05, comp06 y comp20, en las cuales los resultados son menores a los propuestos por [14], pero siguen siendo mejores a los generados por [53].

De manera clara, el algoritmo propuesto es mejor en cada una de las instancias comparadas con la heurística que empleo grafos. Sin embargo, para el caso del recocido simulado, donde los autores emplearon la media para presentar los resultados, el algoritmo también emplea la misma métrica para una comparación justa. De manera general, los resultados obtenidos por el recocido simulado modificado son mejores a los generados por la estrategia, con la única excepción de la instancia comp18, en donde el desempeño del programa fue mejor.

Si se analiza con mayor detenimiento los valores de la instancia comp05 de la Tabla1, el algoritmo en ocasiones obtiene soluciones viables, este comportamiento no es consistente en cada uno de los experimentos.

Si se observa detenidamente la instancia comp05, es posible determinar que es la segunda instancia con mayor número de horas prohibidas para asignar en toda la base de datos. Además, otra característica importante a resaltar es el hecho de ser la segunda instancia mayor número de currícula de toda la base de datos.

Para los diagramas de caja que se muestran a continuación, cabe señalar que para la instancia comp05, los valores tomados para elaborar el gráfico fueron considerando únicamente los valores de las restricciones blandas no satisfechas, siendo esta la única diferencia realizada.

Como se puede observar en las ilustraciones 5, 6, 7 y 8 en general, los resultados obtenidos por el algoritmo se mantienen de manera consistente en cada una de las pruebas. La dispersión de las cajas tiende a ser pequeña en cada una de las instancias, con la excepción de instancia comp05, comp07 y la instancia comp20, en las cuales los resultados pueden variar de manera considerable.

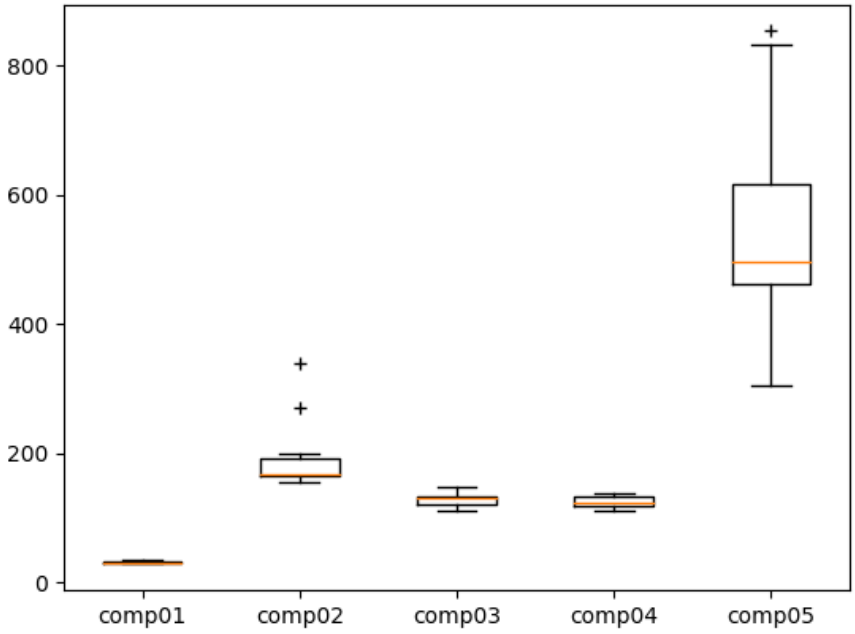

Ilustración 5. Resultados de la instancia comp01 a la comp05.

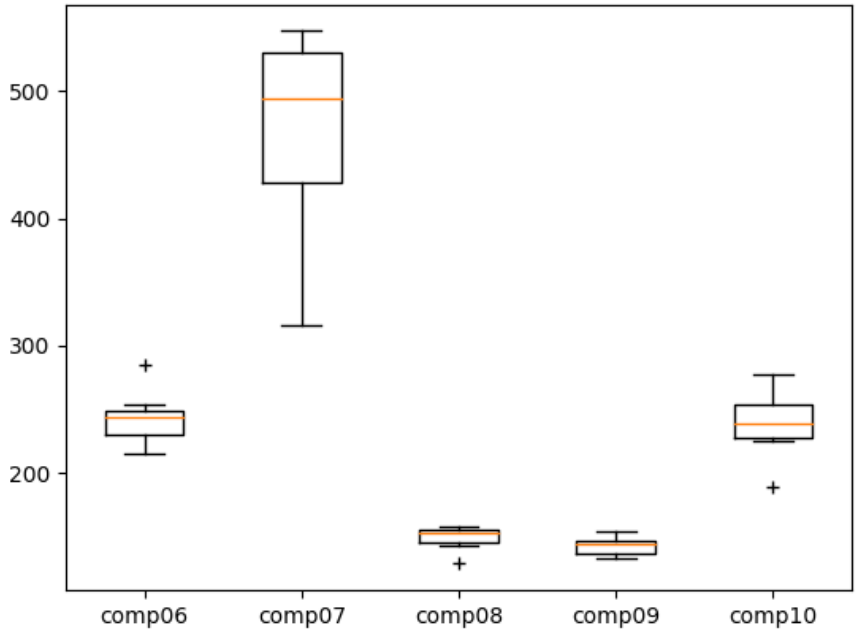

Ilustración 6. Resultados de la instancia comp06 a la comp10.

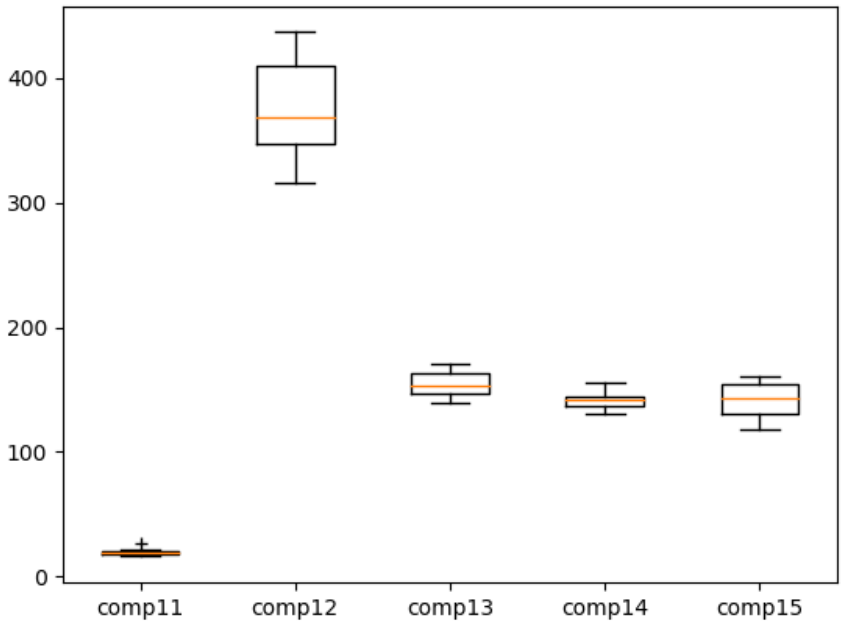

Ilustración 7. Resultados de la instancia comp11 a la comp15.

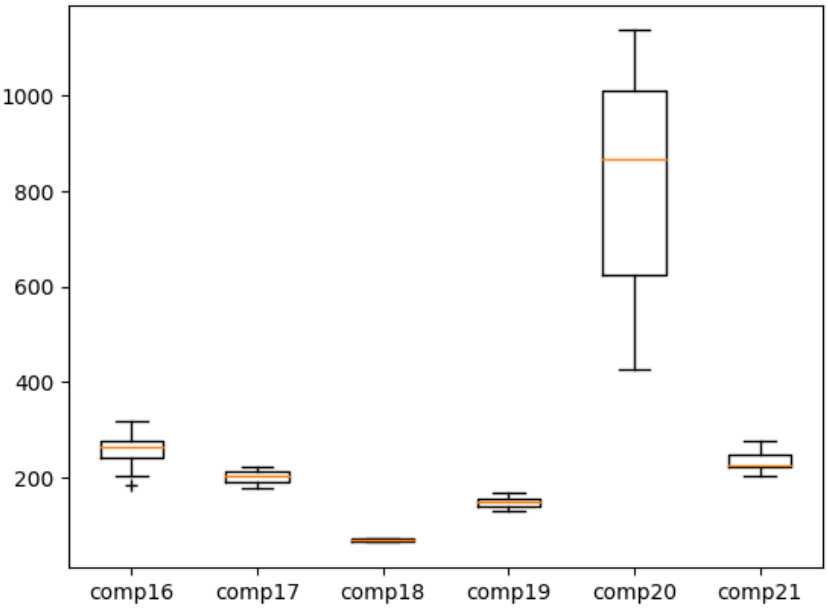

Ilustración 8. Resultados de la instancia comp16 a la comp21.

Adicionalmente se realizaron dos ilustraciones más (ilustración 9 e ilustración 10) en donde se muestra la desviación estándar producida por el algoritmo en cada una de las instancias del problema. Con la intención de analizar puntualmente, la consistencia con la que se obtienen los resultados.

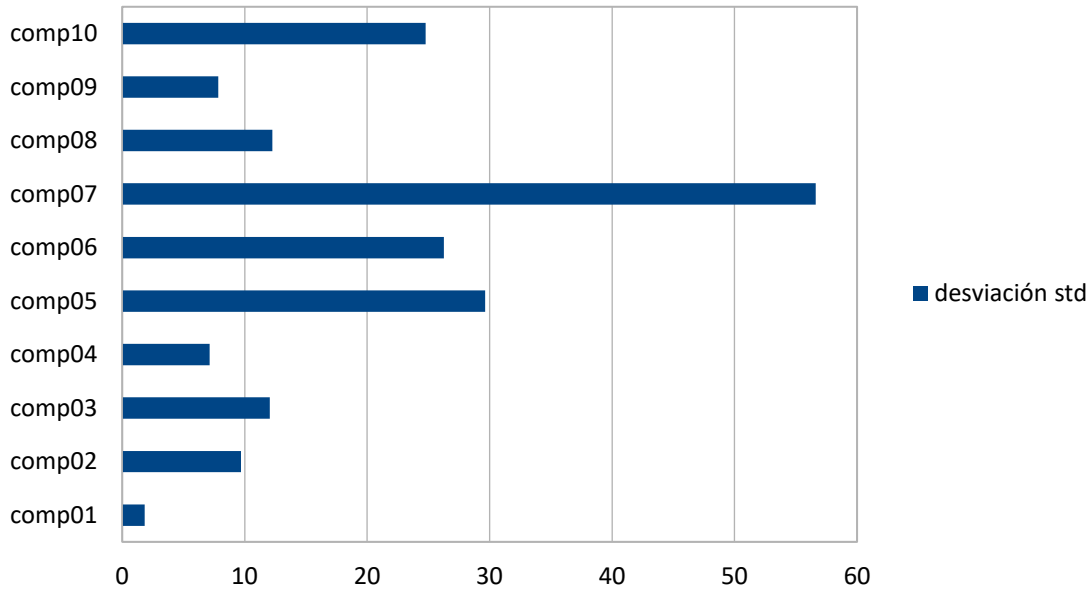

Ilustración 9. Desviación estándar comp01 a comp10.

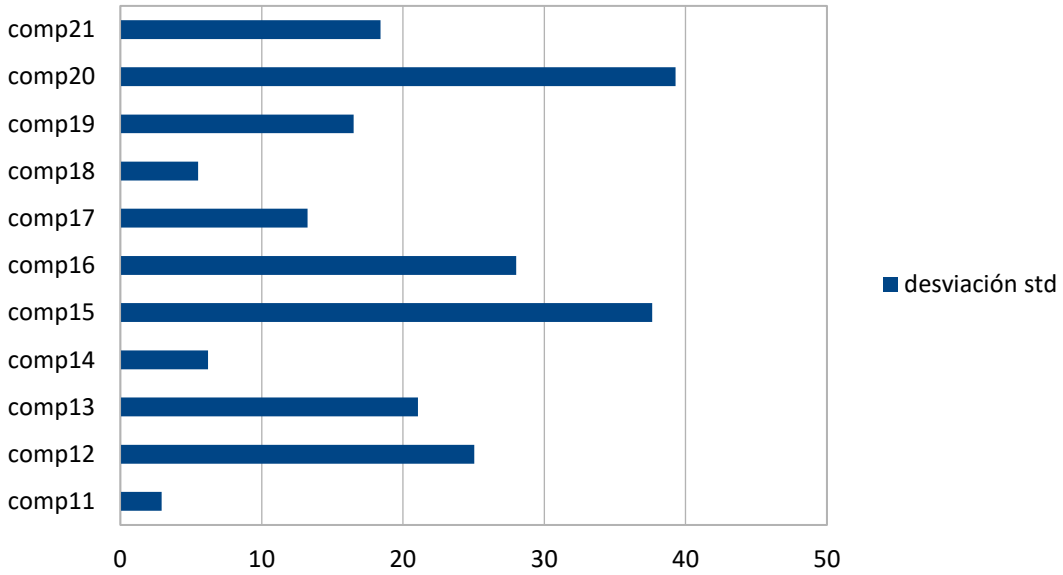

Ilustración 10. Desviación estándar comp11 a comp21.

Como se puede observar, en general, los resultados en las instancias son consistentes, los valores generados utilizando la desviación estándar son adecuados. Con la excepción de la instancia comp07, comp15 y comp20.

Capítulo 8

8. Conclusiones

En el presente trabajo se propuso un algoritmo hiper-heurístico para abordar el problema CBCT, donde se emplea el esquema de ruleta para seleccionar a las heurísticas de bajo nivel y el uso del recocido simulado como técnica que guio el proceso de optimización. Además, el algoritmo comienza con una solución inicial generada con una estrategia que minimiza el número de restricciones duras insatisfechas.

Durante el proceso de búsqueda, el algoritmo mantiene una memoria sobre el desempeño que ha tenido cada heurística de bajo nivel a lo largo del proceso de optimización, con el fin de que pasado un periodo de tiempo, sean evaluadas y modifiquen su participación en los posteriores pasos de la búsqueda.

A mayor desempeño la heurística de bajo nivel tendrá mayor posibilidad de ser seleccionada, caso contrario si el rendimiento es bajo habrá menor probabilidad de ser elegida. Sin embargo, todas las heurísticas tienen una oportunidad mínima de elección, debido a que es posible que en otras etapas de la búsqueda la estrategia tenga una mejor eficiencia.

La propuesta hiper-heurística genera resultados competitivos, siendo un mecanismo clave el esquema de ruleta implementado como estrategia de selección de heurísticas. Debido a que permite participar a todas las heurísticas de bajo nivel en el proceso de optimización, dando mayor tiempo de ejecución a las que proporcionan mejor desempeño en la resolución del problema.

El uso de diferentes mecanismos le permitió al algoritmo explorar diferentes regiones del espacio solución, en conjunto con la inclusión del recocido simulado en el nivel alto del algoritmo permitió evadir los óptimos locales, a la par de que conforme desciende la temperatura, se prioriza la obtención de una buena solución volviendo más refinada la respuesta final.

La elección del valor de penalización por romper una restricción dura tiene un impacto considerable en dos aspectos del programa: los resultados obtenidos y la consistencia de generar soluciones factibles.

A un mayor incremento del factor de penalización, será más difícil para el algoritmo propuesto el encontrar soluciones de costos bajos, debido a que tendrá mayor dificultad para salir de óptimos locales, pero se compensa con mayor estabilidad de finalizar con soluciones factibles.

Para el caso con penalizaciones menores se obtiene soluciones con valores bajos, pero se tienden a que la respuesta final sea inviable por tener restricciones duras insatisfechas. Esto se debe a que, al tener factores bajos de penalización el algoritmo puede tener problemas para converger en una solución factible.

Por lo que, para obtener resultados aceptables y de manera consistente en el algoritmo propuesto, el valor ideal de penalización se encuentra cerca de 300 por romper una restricción dura y 1 por violar una restricción suave.

El desempeño del algoritmo en la mayoría de las pruebas realizadas es estable, en los diagramas de cajas se puede observar como la propuesta obtiene resultados con puntajes consistentes entre cada ejecución, exceptuando tres instancias, por lo que se puede decir que los resultados entre una ejecución y otra tendrán una variación baja.

Sí bien las instancias comp07, comp15 y comp20, no tienen una característica sobresaliente con respecto a otras (e.g. tener el mayor número de horas prohibidas o poseer el mayor número de currículas), posiblemente la combinación de las variables que tienen hace que el algoritmo tenga dificultades en tener resultados consistentes.

Finalmente hay que destacar que, para algunas instancias particulares, el resultado generado se encuentra cercano al mejor punto conocido.

8.1 Trabajo a futuro

Como trabajo futuro se examinarán otros mecanismos de bajo nivel más sofisticados; se ha observado en otros artículos la implementación de heurística de nivel inferior que abordan directamente una restricción del problema.

Adicionalmente se implementarán diferentes metaheurística en el nivel alto que puedan proporcionar un mayor beneficio al obtenido con el recocido simulado. Se podría intentar probar con otros criterios de selección de una heurística, como lo podría ser una función de evaluación donde el tiempo y el desempeño de una heurística de bajo nivel a lo largo de todo el proceso de optimización tenga un papel más relevante.

Además, se plantea continuar con la investigación empleando otros esquemas hiper-heurísticos que puedan ser aplicados al problema. Por mencionar un ejemplo, se podría considerar el uso de heurísticas de bajo nivel cuyo enfoque sea constructivo.

Finalmente, se tiene la intensión de probar el programa con otras bases de datos que no pertenezcan a la ITC. Con el objetivo tener un espectro más amplio del desempeño del algoritmo para el problema CBCT.

Bibliografía

[1] R. Martí, P. M. Pardalos, M. G. C. Resende, “Handbook of Heuristics”, Springer, 2018.

[2] I. Boussaïd, J. Lepagnot, P. Siarry, “A survey on optimization metaheuristics”, Information Sciences, vol. 237, pp. 82–117, 2013.

[3] González-Calero, P.A., Gómez-Martín, M.A. “Arttficial Intelligence for Computer Games” Springer, Universidad Complutense de Madrid, 2011.

[4] J. Mc Carthy, M. L. Minsky, N. Rochester, C. E. Shannon, “A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence”, AI Magazine, 1955.

[5] X. S. Yang, “Recent Advances in Swarm Intelligence and Evolutionary Computation”, Studies in Computational Intelligence, Vol. 585, 2015.

[6] L. Araujo, C. Cervigón, “Algoritmo evolutivos: un enfoque práctico”, RA-MA, México, 2009.

[7] M. M. Drugan, “Reinforcement learning versus evolutionary computation: A survey on hybrid algorithms”,Swarm and Evolutionary Computation, vol. 44, pp. 228–246, 2019.

[8] H. Babaei, J. Karimpour, A. Hadidi, “A survey of approaches for university course timetabling problem”, Computers & Industrial Engineering, Vol. 86, pp 43-59, 2015.

[9] N. Pillay, R. Qu, “Hyper-Heuristics: Theory and Applications”, Springer, Natural Computing Series, 2018.

[10] N. R. Sabar, M. Ayob, G. Kendall, R. Qu, “A Dynamic Multiarmed Bandit-Gene Expression Programming Hyper-Heuristic for Combinatorial Optimization Problems”, European Journal of Operational Research, IEEE Transactions on Cybernetics, Volumen 45, Número 2, pp. 217-228, 2015.

[11] N. R. Sabar, M. Ayob, G. Kendall, R. Qu, “Automatic Design of a Hyper-Heuristic Framework With Gene Expression Programming for Combinatorial Optimization Problems”, IEEE Transactions on Evolutionary Computation, Volumen 19, Número. 3, pp. 309-325, 2015.

[12] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, R. Qu, “Hyper-heuristics: a survey of the state of the art”, Journal of the Operational Research Society, vol. 64, pp. 1695–1724, 2013.

[13] J. A. Soria-Alcaraz, G. Ochoa, M. A. Sotelo-Figeroa, E. K. Burke, “A methodology for determining an effective subset of heuristics in selection hyper-heuristics”, European Journal of Operational Research, Volumen 260, pp. 972-983, 2017.

[14] S. S. Habashi, C. Salama, A. H. Yousef, H. M. A. Fahmy, “Adaptive Diversifying Hyper-Heuristic Based Approach for Timetabling Problems”, IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference, IEMCON 2018, pp. 259-266 2018.

[15] K. Smith-Miles, L. Lopes, “Measuring instance difficulty for combinatorial optimization problems”, Computers & Operations Research, vol. 39, pp. 875–889, 2012.

[16] C. K. Teoh, A. Wibowo, M. S. Ngadiman, “Review of state of the art for metaheuristic techniques in Academic Scheduling Problems”, Artificial Intelligence Review, vol. 44, pp. 1–21, 2015.

[17] Competencia Internacional de Horarios, problema de horarios 2007 y cursos basados en currículum, http://www.cs.qub.ac.uk/itc2007/curriculmcourse/course_curriculm_index_files/problemmodel.htm

[18] M. Lindahl, T. Stidsen, M. Sørensen, “Quality recovering of university timetables”, European Journal of Operational Research, vol. 276, pp. 422–435, 2019.

[19] University of Oxford, definición de Inteligencia Artificial, https://www.lexico.com/definition/artificial_intelligence

[20] C. Blum, J. Puchinger, G. R. Raidl, A. Roli, “Hybrid metaheuristics in combinatorial optimization: A survey”, Applied Soft Computing, vol. 11, pp. 4135–4151, 2011.

[21] Yang X. S. “Nature-inspired metaheuristic algorithms: success and new challenges”, Journal Computer Engineering Information Technologies, vol. 1, pp. 1–3, 2012.

[22] C. A. C. Coello, G. B. Lamont, D. A. V. Veldhuizen, “Evolutionary Algorithms for Solving Multiobjective Problems”, Springer, 2007.

[23] K. Hussain, M. N. M. Salleh, S. Cheng, Y. Shi, “Metaheuristic research: a comprehensive survey”, Artificial Intelligence Review, vol. 52, pp. 2191-233, 2019.

[24] R. Lewis, J. Thompson, “Analysing the effects of solution space connectivity with an effective metaheuristic for the course timetabling problem”, European Journal of Operational Research, vol. 240 pp. 637–648, 2015.

[25] J. F. Franco, E. M. Toro y R. A. Gallego, “Problema de asignación óptima de salones resuelto con Búsqueda Tabú”, Ingeniería & Desarrollo, Número 24, pp. 149-175, 2008.

[26] R. Qu, N. Pham, R. Bai, G. Kendall, “Hybridising heuristics within an estimation distribution algorithm for examination timetabling”, Applied Intelligence, vol. 42, pp. 679–693, 2015.

[27] G. R. Raidl, “Decomposition based hybrid metaheuristics”, European Journal of Operational Research, vol. 244, pp. 66–76, 2015.

[28] V. Kralev, R. Kraleva, “A local search algorithm based on chromatic classes for university course timetabling problem”, International Journal of Advanced Computer Research, Vol 7, 2016.

[29] R. P. Badoni, D.K. Gupta, P. Mishra, “A new hybrid algorithm for university course timetabling problem using events based on groupings of students”, Computers & Industrial Engineering, Volume 78, pp. 12-25, 2014.

[30] T. Song, S. Liu, X. Tang, X. Peng, M. Chem, “An iterated local search algorithm for the University Course Timetabling Problem”, Applied Soft Computing, Volume 68, pp. 597-608, 2018.

[31] L. M. Ortiz, J. M. Carpio, H. J. Puga, C. L. Díaz, C. L. Ramírez y J. A. Soria-Alcaraz, “Comparativa de algoritmos bioinspirados aplicados al problema de calendarización de horarios”, Research in Computing Science 94, pp. 33-43, 2015.

[32] L. W. Shena, H. Asmunia, F. C. Weng, “A Modified Migrating Bird Optimization For University Course Timetabling Problem”, Sciences & Engineering, pp. 89-96, 2015.

[33] Z. Min-Xia, Z. Bei, Q. Neng, “University course timetabling using a new ecogeography-based optimization algorithm”, Natural Computing, vol. 16, pp. 61-74, 2017.

[34] C. Weng Fong, H. Asmuni, W. Shen Lam, B. McCollum, P. McMullan, “A Novel Hybrid Swarm based Approach for Curriculum based Course Timetabling Problem”, IEEE Congress on Evolutionary Computation, pp. 6-11, 2014.

[35] M. Kalender, A. Kheiri, E. Özcan, E. K. Burke, “A greedy gradient-simulated annealing selection hyper-heuristic”, Soft Computing, vol. 17, pp. 2279–2292, 2013.

[36] C. W. Fong, H. Asmuni and B. McCollum, "A Hybrid Swarm-Based Approach to University Timetabling", IEEE Transactions on Evolutionary Computation, vol. 19, no. 6, pp. 870-884, 2015.

[37] J. Wahid, N. Mohd Hussin, “Construction of Initial Solution Population for Curriculum-Based Course Timetabling using Combination of Graph Heuristics”, Journal of Telecommunication, Electronic and Computer Engineering, Vol. 8 No. 8, pp. 91-95, 2017.

[38] J. B. Matias, A. C. Fajardo, R. M. Medina, “Examining Genetic Algorithm with Guided Search and Self-Adaptive Neighborhood Strategies for Curriculum-Based

Course Timetable Problem", IEEE, Fourth International Conference on Advances in Computing, Comunication & Automation (ICACCA), pp. 1-6, 2019.

[39] G. Dueck, "New optimization heuristics: the great deluge algorithm and the record-to-record travel", Journal of Computational Physics. Vol. 104, pp. 86–92, 1993.

[40] M. Alzaqebah, S. Abdullah, "Hybrid bee colony optimization for examination timetabling problems", Computers & Operations Research, vol. 54, pp. 142–154, 2015.

[41] A. L. Bolaji, A. T. Khader, M. A. Al-Betar, M. A. Awadallah, "University course timetabling using hybridized artificial bee colony with hill climbing optimizer", Journal of Computational Science, vol. 5, pp. 809–818, 2014.

[42] J. H. Holland, "Hidden order: how adaptation builds complexity". Addison-Wesley, pp. 1-185, 1995

[43] S. Sivanandam, & S. Deepa, "Introduction to Genetic Algorithms", Springer., 2008.

[44] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, "Optimization by simulated annealing", Science, vol. 220, issue 4598, pp. 671-680, 1983.

[45] A. Kheiri, E. Keedwell, "A Hidden Markov Model Approach to the Problem of Heuristic Selection in Hyper-Heuristics with a Case Study in High School Timetabling Problems", Evolutionary Computation, vol. 25, No. 3, pp. 473–501, 2017.

[46] N. Pillay., "A review of hyper-heuristics for educational timetabling", Annals of Operations Research, vol. 239, pp. 3-38, 2014.

[47] J. A. Soria-Alcaraz, E. Özcan, J. Swan, G. Kendall y M. Carpio, “Iterated local search using an add and delete hyper-heuristic for university course timetabling”, Applied Soft Computing, volumen 40, pp. 581-593, 2016.

[48] L. N. Ahmed, E. Özcan, A. Kheiri, “Solving high school timetabling problems worldwide using selection hyper-heuristics”, Expert Systems with Applications, vol. 42, pp. 5463–5471, 2015.

[49] N. R. Sabar, G. Kendall, “Population based Monte Carlo tree search hyper-heuristic for combinatorial optimization problems”, Information Sciences, vol. 314, pp. 225–239, 2015.

[50] J. P. Rankhambe, R. M. Pandharpatte, “A Survey on Examination Scheduling Problem (ESP) and Hyper-Heuristics Approaches for Solving ESP”, International Conference on Information Processing (ICIP), Vishwakarma Institute of Technology, pp. 16-19, 2015

[51] E. K. Burke, R. Qu, A. Soghier, “Adaptive selection of heuristics for improving exam timetables”, Annals of Operations Research, vol. 218, pp. 129–145, 2014.

[52] A. Kheiri, E. Özcan, A. J. Parkes, “A stochastic local search algorithm with adaptive acceptance for high-school timetabling”, Annals of Operations Research, vol. 239, pp. 135–151, 2016.

[53] Herramienta de desarrollo Hyflex y el desafío de búsqueda de dominio cruzado, http://www.asap.cs.nott.ac.uk/external/chesc2011/hyflex_description.html

[54] Y. Lei, M. Gong, L. Jiao, Y. Zuo, “A memetic algorithm based on hyper-heuristics for examination timetabling problems”, European Journal of Operational Research,

International Journal of Intelligent Computing and Cybernetics, vol. 8 No. 2, pp. 139-151, 2015.

[55] A. Muklason, P. C. Bwananesia, S. Hidayatul, N. D. Angresti, V. Azthanty, “Automated Examination Timetabling Optimization Using Greedy-Late Acceptance-Hyperheuristic Algorithm”, INTERNATIONAL CONFERENCE ON ELECTRICAL ENGINEERING AND COMPUTER SCIENCE (ICECOS), pp. 201-206, 2018.

[55] N. C. Fink Bagger, G. Desaulniers, J. Desrosiers, “Daily course pattern formulation and valid inequalities for the curriculum-based course timetabling problem”, Journal of Scheduling, vol. 22, pp. 155–172, 2019.

[56] M. A. Awadallah, M. A. Al-Betar, A. T. Khader, A. L. Bolaji, M. Alkoffash, “Hybridization of harmony search with hill climbing for highly constrained nurse rostering problem”, Neural Computing and Applications, vol. 28, pp. 463–482, 2017.

[57] N. Pillay, E. Özcan, “Automated generation of constructive ordering heuristics for educational timetabling”, Annals of Operations Research, vol. 275, pp. 181–208, 2019.

[58] N. C. F. Bagger, M. Sørensen, T. R. Stidsen, “Benders’ decomposition for curriculum-based course timetabling”, Computers and Operations Research, vol. 91, pp. 178–189, 2018.

[59] M. Lindahl, A. J. Mason, T. Stidsen, M. Sørensen, “A strategic view of University timetabling”, European Journal of Operational Research, vol. 266, pp. 35–45, 2018.

[60] T. Chong Keat, H. Haron, A. Wibowo, M. S. Ngadiman, “A Heuristic Room Matching Algorithm in Generating Enhanced Initial Seed for the University Course Timetabling Problem”, Research Journal of Applied Sciences, Engineering and Technology, vol. 10, pp. 882-889, 2015

[61] Competencia Internacional de Horarios 2007, http://www.cs.qub.ac.uk/itc2007/index.htm

[62] Competencia Internacional de Horarios 2002, http://sferics.idsia.ch/Files/ttcomp2002/

[63] Competencia Internacional de Horarios 2011, https://www.utwente.nl/en/eemcs/dmmp/hstt/itc2011/

[64] Competencia Internacional de Horarios 2019, https://www.itc2019.org/home

[65] R. Bellio, S. Ceschia, L. Di Gaspero, A. Schaerf, T. Urli, “Feature-based tuning of simulated annealing applied to the curriculum-based course timetabling problem”, Computers & Operations Research vol. 65, pp. 83-92, 2016.

[66] J. Wahid, N. M. Hussin, K. Malaysia, P. Malaysia, “Harmony Great Deluge for Solving Curriculum Based Course Timetabling Problem”, IEEE, 3rd International Conference on System Engineering and Technology, pp. 19–20, 2013.

[67] C. Weng Fong, H. Asmuni, B. McCollum, P. McMullan, S. Omatu, “A new hybrid imperialist swarm-based optimization algorithm for university timetabling problems”, Information Sciences vol. 283, pp. 1–21, 2014.

[68] A. L. Bolaji, A. T. Khader, M. A. Al-Betar,M. A. Awadallah, “University course timetabling using hybridized artificial bee colony with hill climbing optimizer”, Journal of Computational Science, vol. 5, pp. 809–818, 2015.

[69] M. Chiarandini, M. Birattari, K. Socha, O. Rossi-Doria, “An effective hybrid algorithm for university course timetabling”, Journal of Scheduling, vol. 9, pp. 403–432, 2006.

[70] R. Qu, N. Pham, R. Bai, G. Kendall, “Hybridising heuristics within an estimation distribution algorithm for examination timetabling”, Applied Intelligence, vol 42, pp. 679-693, 2015.

[71] S. Muthuraman, V. P. Venkatesan, “A Comprehensive Study on Hybrid Meta-Heuristic Approaches IEEE Used for Solving Combinatorial Optimization Problems”, IEEE, World Congress on Computing and Communication Technologies (WCCCT), pp. 185-190, 2017.

[72] M. Banbara, K. Inoue, B. Kaufmann, T. Okimoto, T. Schaub, T. Soh, N. Tamura, P. Wanko, “teaspoon: solving the curriculum-based course timetabling problems with answer set programming”, Annals of Operations Research, vol. 275, pp. 3-37, 2019.

Anexo: Pseudocódigo

```
ejecutar_HH:
    temp_actual = TEMP_INICIAL
    temp_actual *= ALPHA
    horario_i = generar_solucion()
    horario_j = copiar(horario_j)
    mejor_horario = copiar(horario_i)
    tiempo_actual = tiempo()
    MIENTRAS: tiempo_actual > 500; HACER
        REPETIR N_ITERACIONES
            h_baja = seleccionar_h()
            h_baja.seleccionada += 1
            h_baja.aplicar(horario_j)
            SI horario_j.costo < horario_i.costo O
                    criterio_metropoli(horario_i.costo,
                            horario_j.costo, temp_actual):
                horario_i = copiar(horario_j)
                h_baja.aceptada += 1
                SI horario_j.costo < mejor_horario.costo:
                    mejor_horario = copiar(horario_j)
            DE  OTRO MODO:
                    h_baja.regresar_inicial(horario_j)
        temp_actual *= ALPHA
```

actualizar_ruleta()

tiempo_actual = tiempo()

REGRESAR mejor_horario

Anotaciones:

TEMP_INICIAL: parámetro con el que inicial el recocido simulado.

ALPHA: utilizado para el esquema de enfriamiento del recocido simulado.

N_ITERACIONES: número de iteraciones a realizar durante cada cambio de temperatura.

generar_solucion: método que inicializa un calendario.

copiar: método para realizar una duplicación de la solución

tiempo: método para calcular el tiempo de ejecución del algoritmo.

seleccionar_h: método para elegir una heurística de bajo nivel.

criterio_metropoli: método que es utilizado en algoritmo de recocido simulado y que permite salir de óptimos locales en procesos de optimización.

actualizar_ruleta: método para restablecer los nuevos valores de la ruleta de acuerdo al desempeño generado por cada heurística.

generar_solucion:

horario[SALONES][HORAS]

ordenar_lecturas(lecturas)

PARA lectura EN lecturas:

salon, hora = encontrar_celda(horario, lecturas.r_prohibida)

// r_prohibida, son las horas (en caso de haber) donde no

// es posible asignar la lectura

SI hora != NULO:

horario[salon][hora] = lectura

DE LO CONTRARIO:

salon, hora = buscar_celda_vacia(horario)

horario[salon][hora] = lectura

Anotaciones:

SALONES: el número de habitaciones disponibles para asignar lecturas.

HORAS: el número de horas del que disponemos para alojar a las lecturas de los cursos.

lecturas: todas las lecturas pertenecientes a los cursos.

horario: es una matriz que contiene a las lecturas de todos los horarios.

ordenar_lecturas: método que ordena las lecturas de acuerdo a su dificultad. Por dificultad, se entiende que a menor número de salones viables en donde alojar las lecturas

encontrar_celda: respetando las reglas descritas en el capítulo X, se busca una celda donde se pueda alojar la lectura.

buscar_celda_vacia: busca una celda cualquier vacía.

seleccionar_h:

```
n_random = random()
PARA h EN heuristicas:
        SI n_random < h.segmento:
                REGRESAR h // regresar la heurística perteneciente
                                // segmento
REGRESAR h        // regresar la última heurística
```

Anotaciones:

random: método para generar números aleatorios.

actualizar_ruleta:

```
desempeno = [N_H]
nuevo_segmento = [N_H]
i = 0
PARA h EN heuristicas:
        seleccionada = h.seleccionada
```

```
            aceptada = max(1, h.aceptada)
            desempeno[i] = (aceptada / seleccionada)
            i += 1
        sum_desempeno = max(suma(desempeno), 1)
        acarreo = 0
        i = 0
        PARA h EN heuristicas:
            nuevo_segmento =  desempeno[i] / sum_desempeno
                nuevo_segmento = MIN_PORC + (nuevo_segmento *
                 RESTANTE_PORC)
            h.segmento = nuevo_segmento + acarreo
            acarreo += nuevo_segmento
```

Anotaciones

N_H: número de heurísticas de bajo nivel en el algoritmo.

MIN_PORC: porcentaje mínimo para que una heurística pueda ser seleccionada.

RESTANTE_PORC: porcentaje restante después de haber repartido el porcentaje mínimo entre cada heurística.

heuristicas: arreglo que contiene a las heurísticas de bajo nivel.

max: método para elegir el valor máximo de acuerdo a los argumentos recibidos.

suma: método para sumar los valores en un arreglo recibido.

Casa abierta al tiempo
UNIVERSIDAD AUTÓNOMA METROPOLITANA

ACTA DE EXAMEN DE GRADO

No. 00091
Matrícula: 2183802139

Asignación de salones por medio de una hiper-heurística.

Con base en la Legislación de la Universidad Autónoma Metropolitana, en la Ciudad de México se presentaron a las 13:00 horas del día 26 del mes de marzo del año 2021 POR VÍA REMOTA ELECTRÓNICA, los suscritos miembros del jurado designado por la Comisión del Posgrado:

DR. JORGE ALBERTO SORIA ALCARAZ
M. EN C. ALMA EDITH MARTINEZ LICONA
DR. ERIC ALFREDO RINCON GARCIA

Bajo la Presidencia del primero y con carácter de Secretario el último, se reunieron para proceder al Examen de Grado cuya denominación aparece al margen, para la obtención del grado de:

MAESTRO EN CIENCIAS (CIENCIAS Y TECNOLOGIAS DE LA INFORMACION)

DE: JORGE CARLOS GONZALEZ GONZALEZ

y de acuerdo con el artículo 78 fracción III del Reglamento de Estudios Superiores de la Universidad Autónoma Metropolitana, los miembros del jurado resolvieron:

APROBAR

Acto continuo, el presidente del jurado comunicó al interesado el resultado de la evaluación y, en caso aprobatorio, le fue tomada la protesta.

UNIVERSIDAD AUTONOMA METROPOLITANA
Casa abierta al tiempo
DIRECCION DE SISTEMAS ESCOLARES

JORGE CARLOS GONZALEZ GONZALEZ
ALUMNO

REVISÓ

MTRA. ROSALIA SERRANO DE LA PAZ
DIRECTORA DE SISTEMAS ESCOLARES

DIRECTOR DE LA DIVISIÓN DE CBI

DR. JESUS ALBERTO OCHOA TAPIA

PRESIDENTE

DR. JORGE ALBERTO SORIA ALCARAZ

VOCAL

M. EN C. ALMA EDITH MARTINEZ LICONA

SECRETARIO

DR. ERIC ALFREDO RINCON GARCIA

El presente documento cuenta con la firma –autógrafa, escaneada o digital, según corresponda- del funcionario universitario competente, que certifica que las firmas que aparecen en esta acta – Temporal, digital o dictamen- son auténticas y las mismas que usan los c.c. profesores mencionados en ella