Solución del problema de programación de cursos universitarios (UCTP) utilizando un método híbrido basado en Algoritmos Genéticos.

Andrés José Mora Esquivel

Trabajo de grado para optar el título de Ingeniero Industrial

Director

Javier Eduardo Arias Osorio

Magíster en Administración

Universidad Industrial de Santander

Facultad de Ingenierías Físico Mecánicas

Escuela de Estudios Industriales y Empresariales

Bucaramanga

2019

Agradecimientos

A Dios, por permitirme vivir esta experiencia y guiarme durante estos años.

A mis padres, Myriam y Jose, por haber estado acompañándome y apoyándome durante ente camino.

A mis profesores, pues su labor y dedicación han aportado enormemente a mi formación como persona y profesional.

A mi alma mater, por brindarme los medios necesarios para cumplir esta meta.

A mis amigos y compañeros, quienes me acompañaron en este camino y han sido apoyo en todo momento.

Al grupo de investigación OPALO, el cual ha sido de gran apoyo para llevar a cabo este proyecto.

Al profesor Javier Eduardo Arias, por su paciencia y orientación a lo largo de este trabajo.

Andrés José Mora Esquivel

Tabla de Contenido

Lista de tablas

Lista de figuras

Lista de apéndices

(Ver apéndices adjuntos en el CD y pueden visualizarlos en la Base de Datos de la Biblioteca UIS)

RESUMEN

Título: Solución del problema de programación de cursos universitarios (UCTP) utilizando un método híbrido basado en Algoritmos Genéticos. *

Autor: Andrés José Mora Esquivel**

Palabras clave: Programación de cursos universitarios, Metaheurísticas, Programación Lineal, HGATS.

Descripción:

En la presente investigación se ataca el problema de programación de cursos universitarios, encontrado en la literatura como University Course Timetabling Problem, el cual es considerado un problema NP-hard, debido a la alta demanda computacional que requiere. Semestralmente, la Escuela de Estudios Industriales y Empresariales de la Universidad Industrial de Santander lleva a cabo la Programación de asignaturas para el Programa de Ingeniería Industrial, considerando los profesores, salones y franjas disponibles a lo largo de una semana.

Para dar solución al problema, se propone un modelo de Programación Lineal Entera Mixta que sirve de referencia al momento de dimensionar el problema y las restricciones que deben ser consideradas. Seguidamente, se diseña un método metaheurístico hibrido basado en el algoritmo HGATS, desarrollado por Yang y Jat (2011), el cual combina la capacidad de diversificación del Algoritmo Genético con la estrategia de intensificación del Algoritmo de Búsqueda Tabú. Adicionalmente, se lleva a cabo un diseño factorial fraccionado con el fin de determinar el grado de influencia de los parámetros sobre la calidad de la solución final. Finalmente, se realiza la validación del algoritmo propuesto usando datos correspondientes a la programación de asignaturas de los periodos académicos 2018-1 y 2018-2 para el programa académico de Ingeniería Industrial, obteniendo soluciones aceptables en un tiempo computacional razonable.

* Proyecto de grado
** Facultad de Ingenierías Físico Mecánicas. Escuela de Estudios Industriales y Empresariales. Programa de Ingeniería Industrial. Director: Magister. Javier Eduardo Arias Osorio

ABSTRACT

Title: Solution of University Course Timetabling Problem (UCTP) using a hybrid method based on Genetic Algorithms. *

Author: Andrés José Mora Esquivel**

Keywords: University Course Timetabling Problem, Metaheuristics, Linear Programming, HGATS.

Descripción:

In the present investigation, the problem of programming university courses, found in the literature as University Course Timetabling Problem, which is considered an NP-hard problem due to the high computational demand it requires, is attacked. Every six months, the School of Industrial and Business Studies of the Industrial University of Santander carries out the Programming of subjects for the Industrial Engineering Program, considering the professors, classrooms and time slots available during a week.

To solve the problem, a Mixed Integer Linear Programming model is proposed that serves as a reference to size the problem and the restrictions that must be considered. Next, a hybrid metaheuristic method is designed based on the HGATS algorithm, developed by Yang and Jat (2011), which combines the diversification capacity of the Genetic Algorithm with the strategy of intensification of the Tabu Search Algorithm. Additionally, a fractional factorial design is carried out in order to determine the degree of influence of the parameters on the quality of the final solution. Finally, the validation of the proposed algorithm is performed using data corresponding to the programming of subjects from academic periods 2018-1 and 2018-2 for the academic program of Industrial Engineering, obtaining acceptable solutions in a reasonable computational time.

*Bachelor Thesis

** Facultad de Ingenierías Físico Mecánicas. Escuela de Estudios Industriales y Empresariales. Programa de Ingeniería Industrial. Director: Magister. Javier Eduardo Arias Osorio

Introducción

La programación de cursos universitarios, más conocida por sus siglas en inglés como UCTP, es una actividad que le permite a las instituciones de educación superior cumplir satisfactoriamente con sus funciones misionales. Ésta es una labor demandante, con un alto grado de dificultad que, si no se hace adecuadamente puede comprometer el proceso de formación de los estudiantes. Un horario defectuoso puede generar desde malestar en miembros de la comunidad universitaria (docentes, administrativos y estudiantes) hasta el desperdicio de recursos (tiempo, planta física y personal). Debido a esto, profesionales e investigadores han estudiado la dinámica de este problema durante los últimos 50 años.

El UCTP es un problema de optimización combinatoria del tipo NP-hard (Cruz, Flores, Martínez, Moreno, y Cruz, 2016). En este problema un conjunto de cursos debe ser asignado a un conjunto de salones durante un periodo de tiempo, considerando una serie de restricciones predefinidas por la institución. Los requerimientos y necesidades de cada universidad ocasionan que exista una gran cantidad de variantes del UCTP, incluso en universidades del mismo país. Dicho esto, diferentes métodos han sido utilizados para abordar problemas horarios de universidades en todo el mundo.

En el siguiente documento se ataca el problema de programación de cursos universitarios en el programa de Ingeniería Industrial de la Universidad Industrial de Santander; para ello, se realizó una revisión preliminar de literatura en la que se encontró que los métodos exactos no son adecuados para resolver dicho problema e inclina la investigación hacia la aplicación de métodos

metaheurísticos. Debido a esto, el propósito de este trabajo es implementar un método metaheurístico que sea capaz de generar horarios factibles que cumplan con las necesidades y requerimientos definidos por los miembros involucrados en la planeación del horario.

Tabla 1.
Cumplimiento de objetivos

Objetivo	**Cumplimiento**
Realizar una revisión de literatura del problema programación de cursos universitarios (UCTP).	Capítulo 2 Apéndice A
Formular un modelo de programación matemática para la solución del problema UCTP, caso EEIE.	Capítulo 3
Desarrollar la metaheurística HGATS para el problema UCTP utilizando el software MATLAB.	Capítulo 4 Apéndice C
Evaluar los resultados obtenidos de la metaheurística para el caso EEIE.	Sección 5.2
Elaborar un artículo de carácter publicable con los resultados de la investigación.	Apéndice I

1. Generalidades del proyecto

1.1 Planteamiento del problema

La generación de horarios en universidades es una tarea que debe realizarse semestralmente con el objetivo de organizar la manera en que se llevaran a cabo las actividades de los diferentes programas académicos, esta labor ,generalmente, está a cargo del personal administrativo de las instituciones e implica un gran esfuerzo debido a la gran cantidad de factores que deben

considerarse (Bucco y Bandeira, 2017). En algunos casos, la generación de los horarios se hace de forma manual, donde la asignación se hace tomando como base los horarios del semestre anterior y haciendo pequeñas modificaciones para ajustarlos a las nuevas necesidades, este procedimiento de ajuste puede ocasionar que algunas actividades entren en conflicto, ya sea debido a que no se cuenten con los recursos (infraestructura, docentes) para asignarlas o se crucen con otras actividades.

El problema de programación de cursos universitarios (UCTP) se centra en la programación de los cursos o materias definidas dentro de un programa académico. El UCTP consiste en la asignación de un conjunto de eventos a un espacio físico en un periodo; los eventos representan todas las actividades que deben programarse según el plan de cada programa académico; el espacio físico hace referencia a los salones, laboratorios, auditorios o infraestructura necesaria para llevar a cabo cada actividad; y el periodo indica cada una de las franjas horarias de las que se dispone para realizar cada actividad, las franjas horarias generalmente se manejan dentro de espacios de tiempo de una semana. Cada evento asignado debe ser atendido por un docente y un grupo de estudiantes considerando que no pueden existir conflictos entre eventos, es decir, ningún grupo de estudiantes o profesor puede ser asignado a dos eventos que se lleven a cabo en un mismo periodo de tiempo.

Al momento de llevar a cabo la programación de los cursos es necesario establecer qué tipo de horario se desea construir, esto depende de las necesidades y requerimientos que los miembros involucrados en la planeación del horario consideren importantes. Un horario puede ser bueno o malo dependiendo de los objetivos que los encargados de la planeación hayan definido.

En la literatura es posible encontrar diversos objetivos planteados por diferentes autores, por ejemplo: maximizar la utilización de los recursos, satisfacer las preferencias del cuerpo docente, generar horarios compactos para los estudiantes, minimizar los cambios de salón para cada curso, minimizar el tiempo de transporte entre edificios, entre otros. Generalmente, estos objetivos están asociados a una función objetivo y se definen como las restricciones suaves del problema, estas restricciones pueden ser violadas y cuantifican la calidad de un horario. Por otra parte, existe un conjunto de restricciones que son conocidas como duras, este tipo de restricciones deben ser cumplidas a cabalidad, una violación en una de estas restricciones ocasiona que el horario sea descartado pues no es aplicable al mundo real. Existen algunas restricciones duras comunes en la mayoría de trabajos, por ejemplo: todos los eventos de cada curso deben ser asignados, un salón solo puede albergar como mucho un solo evento en un periodo, un profesor no puede atender dos eventos al mismo tiempo, un grupo de estudiantes no puede atender más de un evento en el mismo periodo, etc. Cabe destacar que dependiendo del enfoque del autor algunas restricciones duras pueden considerarse como suaves y viceversa, incluso pueden definir restricciones específicas para una universidad como es el caso de Schimmelpfeng y Helber (2007). Pongcharoen, Promtet, Yenradee y Hicks (2008) hacen una revisión de un conjunto de restricciones duras y restricciones suaves que han sido consideradas en trabajos previos.

Considerando lo anteriormente descrito, este proyecto busca abordar el Problema de Programación de Cursos Universitarios en el programa de Ingeniería Industrial de la Escuela de Estudios Industriales y Empresariales de la Universidad Industrial de Santander con el fin de facilitar esta labor administrativa.

La construcción de los horarios del programa está asignada al Coordinador Académico de turno, quien tiene la facultad de usar el método que considere adecuado. Actualmente, la asignación se realiza de forma manual con un procedimiento semi-estructurado que involucra técnicas de ensayo y error. Esto da como resultado programaciones ineficientes y reprocesos innecesarios que implican la utilización de más recursos y retrasan la programación de asignaturas, situación que finalmente no permite encontrar una solución óptima al problema.

1.2 Objetivos

1.2.1 Objetivo general

Diseñar un modelo de optimización para el problema UCTP y solucionarlo utilizando un método híbrido basado en algoritmos genéticos. Caso Escuela de estudios industriales y empresariales (EEIE).

1.2.2 Objetivos específicos

Realizar una revisión de literatura del problema programación de cursos universitarios (UCTP).

Formular un modelo de programación matemática para la solución del problema UCTP, caso EEIE.

Desarrollar la metaheurística HGATS para el problema UCTP utilizando el software MATLAB.

Evaluar los resultados obtenidos de la metaheurística para el caso EEIE.

Elaborar un artículo de carácter publicable con los resultados de la investigación.

1.3 Marco teórico

A continuación, se presentan varios temas de relevancia relacionados con el proyecto.

1.3.1 **Optimización combinatoria.** La optimización combinatoria es un campo de matemáticas aplicadas que combina técnicas de programación lineal combinatoria y la teoría de algoritmos para resolver problemas de optimización sobre estructuras discretas (Cook, Cunningham, Pulleyblank, y Schrijver, 1998).

Blum y Roli (2003) definen un problema de optimización combinatoria como:

Sea $P = (S, f)$ un problema de optimización combinatoria

- Un conjunto de variables $X = \{x_1, \dots, x_n\}$;
- Dominio de las variables $D_1, \dots, D_n$;
- Restricciones entre variables;
- Una $función\ objetivo\ f$ a ser minimizada, donde $f: D_1 \times \dots \times D_n \rightarrow \mathbb{R}^+$;

El conjunto de todas las posibles asignaciones factibles es:

$$S = \{s = \{(x_1, v_1), \dots, (x_n, v_n)\} \mid v_i \in D_i, s \text{ satisface todas las restricciones}\}$$

S es el espacio de búsqueda o espacio de solución, donde cada elemento del conjunto puede ser visto como una solución candidata. Para resolver un problema de optimización combinatoria, debe encontrarse una solución $s^* \in S$ con el mínimo valor de función objetivo, $f(s^*) \leq f(s)\ \forall s \in S$. s^* es llamado una solución óptima global de (S, f) y el conjunto $S^* \subseteq S$ es llamado el conjunto de soluciones óptimas globales.

Algunos ejemplos de problemas de optimización combinatoria son el problema del Vendedor Viajero (TSP), el problema de Asignación Cuadrática (QAP), Horarios y problemas de Programación.

1.3.2 **Complejidad computacional.** Dentro de la teoría de la computación, una de las áreas fundamentales es la teoría de la complejidad, este tiene por finalidad la creación de mecanismos y herramientas capaces de describir y analizar la complejidad de un algoritmo y la complejidad intrínseca de un problema. Para ello, considera 2 tipos de recursos requeridos durante el cómputo para resolver el problema:

- Tiempo: número de pasos de ejecución de un algoritmo para resolver un problema.
- Espacio: Cantidad de memoria computacional utilizada para resolver el problema.

La complejidad de un algoritmo se expresa como una función del tamaño de la entrada del problema, $T(n)$: ratio de tiempo de ejecución y $E(n)$: ratio de espacio de almacenamiento necesario.

Según Sanchis, Ledezma, Iglesias, García, y Alosnso (s. f.) para determinar el nivel de complejidad de un problema de decisión se utiliza una Máquina de Turing (Determinista o No Determinista) y los clasifica como:

Clase P: Contiene aquellos problemas de decisión que una Máquina de Turing Determinista puede resolver en tiempo polinómico. Los problemas de complejidad polinómica son tratables, es decir en la práctica pueden resolverse en un tiempo razonable.

Clase NP: Conjunto de problemas de decisión que una Maquina de Turing No Determinista puede resolver en un tiempo polinómico. Se caracterizan por la existencia de un algoritmo capaz de verificar su solución. El tiempo de computo que se requiere para resolver este tipo de problemas se incrementa conforme crece el tamaño del problema presentando una dependencia funcional que

no admite ser acotada por un polinomio. Los problemas NP pueden dividirse en problemas NP-complete y NP-hard.

El problema NP-complete se caracteriza por ser un problema NP, todos los problemas NP pueden reducirse a un NP-complete en tiempo polinómico.

El problema NP- hard no tiene un algoritmo polinómico que lo resuelva, es así como se utilizan algoritmos que ofrezcan una solución cercana a la óptima.

1.3.3 **Timetabling.** La programación de horarios o calendarización es un tipo de problema de asignación, donde cada problema tiene sus propias características únicas que hacen que difiera de una organización a otra. En la actualidad donde es crucial evitar el desperdicio de tiempo y recursos, la programación de horarios de actividades requiere que los recursos estén en el lugar y momento correcto y en las cantidades adecuadas con el objetivo de operar de forma efectiva y eficiente (Obit, 2010).

En la literatura es posible encontrar variantes del problema aplicadas en diferentes como educación, transporte, deportes, salud, empleo, etc. Algunos de ellos se describen a continuación:

Transport timetabling: en este tipo de problema se desea elaborar una programación de horas para los arribos y llegada de los vehículos (ya sean aéreos o terrestres). Además, se busca encontrar una distribución de asignación de conductores para la operación de transporte con el objetivo de reducir al mínimo el costo del programa operativo.

Sport timetabling: En este tipo de problema se desea elaborar horarios para un evento deportivo, distribuyendo los horarios de los partidos (según las reglas propuestas).

Rostering: hace referencia a la programación de horarios de enfermeras, este implica la asignación de un número específico de enfermeras a un conjunto periodos según las habilidades especificas necesarias en ese periodo.

Employee timetabling: En este tipo de problema se desea elaborar horarios de turno de trabajo para diversas áreas de profesionales. El objetivo de esta variante es poder cumplir las exigencias que requiere cada institución como las horas de trabajo de cada empleado. Además, se debe tener en cuenta las restricciones legales que el ámbito laboral requiera.

Educational timetabling: En este tipo de problema se desea elaborar la asignación de asignaturas en instituciones educativas, tomando en cuenta diversos factores que hacen que el problema tenga una complejidad alta. El objetivo de esta variante es encontrar una distribución óptima que tome en cuenta la disponibilidad de los profesores, las restricciones de cada curso, la disponibilidad y capacidad de las aulas, entre otros. Este problema está sujeto a restricciones fuertes o duras las cuales deben ser cumplidas para construir una distribución de horarios útiles y restricciones suaves que no necesariamente deben ser cumplidas, pero aportan calidad a la solución.

Generalmente, a nivel de educación según Schaerf (1999) estos problemas pueden ser clasificados en tres clases, las cuales son:

School timetabling: un horario escolar generalmente sigue un ciclo cada semana para todas las clases y el objetivo es evitar que los maestros tengan que asistir a dos clases al mismo tiempo. En los horarios escolares, los estudiantes normalmente están pre asignados, solo los profesores y las salas deben asignarse en el problema. En este caso las restricciones duras básicas que deben cumplirse en el horario escolar son: ningún maestro debe ser asignado a más de una clase en el

mismo intervalo de tiempo; tener en cuenta la disponibilidad del maestro para cada intervalo de tiempo; y asignar el número correcto de intervalos de tiempo para cada par de clase docente.

Examination timetabling: un conjunto de exámenes universitarios debe programarse en un número limitado de intervalos de tiempo (períodos) evitando los casos en que los estudiantes tomen más de un examen en el mismo intervalo de tiempo. Cada sala tiene una cierta capacidad que no debe ser violada al asignarle un examen. Estas restricciones se conocen como duras porque son inflexibles. Además de las restricciones duras, generalmente hay varias limitaciones suaves que se consideran deseables, pero no obligatorias. Obviamente, existen diferencias significativas entre las instituciones en cuanto a las restricciones que consideran obligatorias y las que no (Pongcharoen *et al*., 2008). Ejemplos de restricciones blandas comunes son cuando los estudiantes prefieren distribuir los exámenes tanto como sea posible durante la semana de exámenes, o la institución quiere programar exámenes extensos antes (para dar más tiempo para resolverlos), algunos miembros del personal de la institución también pueden tener preferencias específicas, por ejemplo, con respecto a los deberes de vigilancia.

University Course timetabling: en el horario de cursos universitarios se hace la programación semanal de todas las clases de un conjunto de cursos universitarios, evitando que clases que tienen estudiantes en común se asignen al mismo intervalo de tiempo. Por lo tanto, la programación de cursos universitarios es el proceso de asignar, sujeto a restricciones, salas limitadas y rangos de tiempo para que tenga lugar un conjunto de cursos. Usualmente, además de construir un horario factible (cuando todas las restricciones duras son satisfechas), existen metas deseables como minimizar el número de asignaciones indeseables. Tales objetivos deseables generalmente se expresan como restricciones suaves.

La figura 1 presenta de forma resumida la forma en que se divide el problema de programación de horarios.

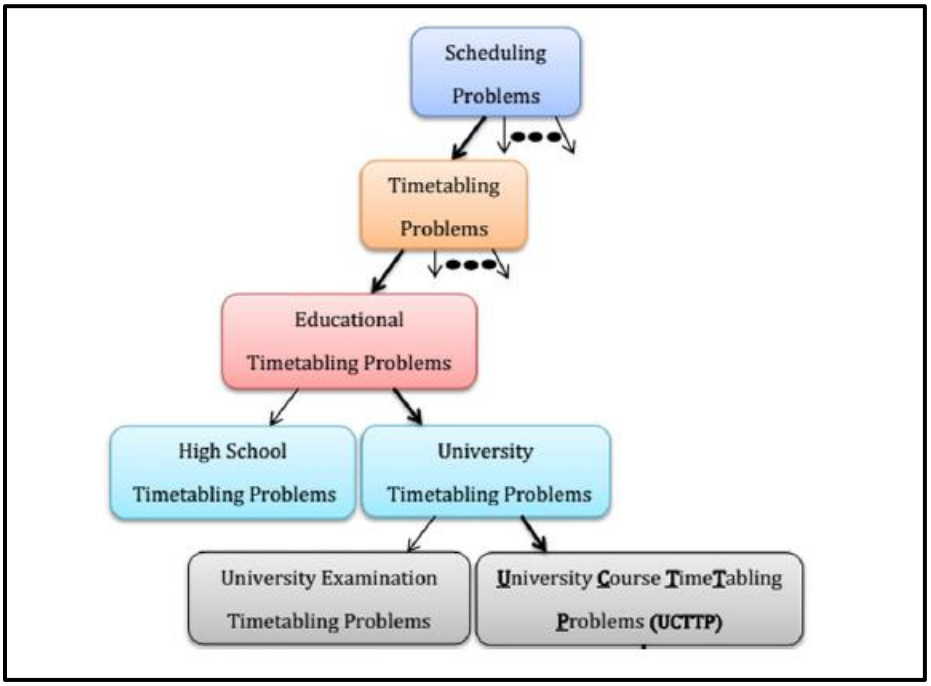

Figura 1. Diagrama del problema de programación de cursos universitarios. Adaptado de Babaei, Karimpour, y Hadidi (2015). Recuperado de http://dx.doi.org/10.1016/j.cie.2014.11.010

El presente proyecto se centra en el estudio de la programación de cursos universitarios, en el siguiente apartado se definen algunos detalles relacionados al problema en cuestión.

1.3.4 **University course timetabling problem (UCTP).** El problema de programación de cursos universitarios ha sido clasificado como un problema de NP-hard, para el cual es difícil encontrar una solución óptima dentro de un tiempo razonable. Encontrar soluciones de buena calidad a estos problemas depende de la técnica en sí misma y de la estructura empleada durante la búsqueda.

Según Abuhamdah y Ayob (2010) en general, los problemas de programación de cursos universitarios implican asignar un conjunto de cursos (eventos), profesores y estudiantes a un número fijo de franjas de tiempo y salones sujetos a diversas restricciones. Las restricciones de este problema pueden clasificarse como duras y blandas. El objetivo del establecimiento de horarios es cumplir todas las restricciones duras e intentar satisfacer las restricciones blandas tanto como sea posible (para producir un horario de alta calidad). Un horario puede representarse como una matriz bidimensional donde las filas corresponden a los salones y las columnas representan las franjas horarias, cada par (salón-franja) es un espacio en el que se puede asignar un curso y un profesor. La figura 2 presenta el esquema de un horario genérico.

	FRANJA							
SALON	1	2	3	4	5	6	. . .	Fp
1	(C1,P1)	(C1,P1)						
2		(C3,P5)	(C3,P5)		(C9,P1)	(C9,P1)		(C2,P2)
3								(C4,P6)
4				(C8,P3)	(C8,P3)			
5		(C6,P9)	(C6,P9)	(C6,P9)				
. . .								
Sk	(C5,P3)	(C5,P3)	(C5,P3)					

Figura 2. Representación matricial de un horario

A continuación, se definen algunos elementos comunes encontrados en la literatura que son necesarios para atacar el problema:

- Cursos: corresponden a cada una de las materias involucradas en el horario, cada curso tiene una intensidad horaria que debe ser programada semanalmente.

- Eventos: es el elemento básico en la construcción de cualquier horario, la duración de un evento regularmente es de una hora. Si un curso tiene una intensidad semanal de cuatro horas, entonces ese curso tiene cuatro eventos que deben ser programados en el horario.
- Currículos: un currículo representa un conjunto de cursos que son atendidos por el mismo grupo de estudiantes, los eventos pertenecientes a un currículo no pueden cruzarse entre sí.
- Profesores: corresponde al personal de tiempo completo o parcial contratado por la institución, cada profesor está capacitado para impartir uno o varios cursos.
- Estudiantes: Corresponde al grupo de personas con adscritos a la institución, pertenecen a un programa académico y tienen la necesidad de asistir a un conjunto de cursos.
- Salones: es el espacio físico necesario para impartir un curso, cada salón posee ciertos atributos que permiten que los eventos puedan ser asignados, así como también una capacidad, generalmente medida como el número de asientos disponibles.
- Días: corresponden a los días de la semana disponibles para hacer la programación, generalmente, son cinco días a la semana de lunes a viernes.
- Periodos: un periodo corresponde a un intervalo de tiempo disponible durante un día, su duración debe ser consecuente con la duración de los eventos.
- Franjas horarias: una franja está constituida por la combinación de un día y un periodo, las franjas totales de la semana corresponden a la multiplicación del total de periodos por el número de días.

Respecto a las restricciones, estas se clasifican en dos tipos, por un lado, existen las restricciones duras, las cuales deben cumplirse para considerar un horario como factible, mientras que, por otra

parte, existen las restricciones suaves, que pueden ser violadas considerando una penalidad y están asociadas a una función objetivo que mide la calidad del horario. Algunas restricciones duras encontradas en la literatura son:

- Sesiones: todos los eventos de cada curso deben ser programados y ser asignados en diferentes franjas
- Conflictos: eventos de cursos que están asignados a un mismo profesor o están incluidos en el mismo currículo no pueden cruzarse entre sí y deben ser programados en diferentes franjas.
- Salones ocupados: un salón solo puede albergar un evento durante un periodo.
- Idoneidad del salón: los eventos solo pueden ser programados en salones que cumplan con los requerimientos establecidos para cada evento.
- Periodos consecutivos: eventos de un mismo curso que son asignados en el mismo día, deben ser asignados en periodos consecutivos.
- Paralelismo: existen casos en los que se requiere que algunos eventos deban ser programados en la misma franja.
- Secuencia: existen eventos que deben programarse en un orden específico.

Respecto a las restricciones suaves, en la literatura se han encontrado las siguientes:

- Satisfacer preferencias: los docentes presentan predilección por impartir sus clases en ciertas franjas horarias.
- Currículo compacto: se desea que los eventos de un mismo currículo que se programen en un día se den de forma consecutiva.

- Estabilidad de salones: se busca que los eventos de un mismo curso o asignados a un mismo profesor sean programados en el mismo salón
- Mínimos días de trabajo: busca que los eventos de un curso estén distribuidos en un mínimo de días diferentes.
- Distancias: existen casos en los que se desea minimizar el tiempo de traslado de un edificio a otro para estudiantes o profesores.

Dada la gran cantidad de variantes del UCTP no es posible definir un problema que incorpore todas las necesidades encontradas en la literatura, por ello, en la mayoría de trabajos se opta por abordar un problema de referencia o el caso específico de una universidad.

1.3.5 **Métodos de solución del UCTP.** En términos de computación, los problemas de generación de horarios a menudo se modelan como problemas de optimización combinatoria (COP). El objetivo general en un COP es encontrar una asignación de valores discretos a las variables (por ejemplo, intervalos de tiempo para cada uno de los eventos que deben programarse) para que la solución sea óptima de acuerdo con algunos criterios (Lewis, 2007). Las técnicas utilizadas de forma general se pueden clasificar en dos grupos: algoritmos exactos y algoritmos aproximados.

1.3.5.1 ***Algoritmos exactos.*** Los algoritmos exactos son métodos que, debido a su estrategia de búsqueda, exploran todo el espacio de soluciones, pudiendo demostrar la optimalizad de una solución. Sin embargo, debido a las tasas de crecimiento exponencial de los espacios de búsqueda para los problemas de UCTP, su aplicación a menudo solo será adecuada para instancias de problema muy pequeñas. Algunos algoritmos exactos se mencionan a continuación:

Método Simplex: es un procesamiento desarrollado por George Dantzing en 1947, para dar soluciones numéricas a problemas de programación lineal. Es un algoritmo iterativo que permite mejorar la solución con cada paso sucesivo; se parte de una solución básica inicial para la función objetivo en un vértice cualquiera, el método consiste en buscar sucesivamente otro vértice que mejore la anterior solución. La búsqueda se hace siempre a través de los lados del polígono de soluciones factibles o de las aristas de la región solución. Finalmente, el algoritmo termina cuando no se puede seguir mejorando más la solución (Hillier y Lieberman, 2010).

Branch and Bound: Este algoritmo es una de las herramientas más utilizadas a la hora de resolver problemas de optimización discreta de tipo NP-hard. El método parte de un conjunto se soluciones candidatas del espacio de búsqueda, estas soluciones constituyen un árbol que inicialmente estaba compuesto solamente por una raíz, en cada iteración un nodo es evaluado construyendo ramas en los nodos que tienen mejores soluciones que el nodo actual (Clausen, 1999).

Otros algoritmos exactos han sido utilizados para abordar el problema de programación de cursos universitarios, entre ellos destacan: Cutting plane algorithm y Branch and cut algorithm (Avella y Vasil'ev, 2005; Burke *et al*.,2012).

1.3.5.2 ***Algoritmos Aproximados.*** Dadas las limitaciones de los algoritmos exactos para encontrar soluciones en los problemas tipo NP-hard, los investigadores han definido una serie de métodos que, sin garantizar el óptimo global, logran alcanzar buenas soluciones a problemas de elevada complejidad en tiempos de cómputo razonables. Los algoritmos aproximados pueden clasificarse en dos grupos: heurísticas y metaheurísticas.

1.3.5.2.1 *Heurísticas:* "Una heurística es una técnica de búsqueda directa que utiliza reglas favorables prácticas para localizar soluciones mejoradas. La ventaja de la heurística es que en general determina buenas soluciones con rapidez utilizando reglas de solución simples" (Taha, 2012, p. 336).

Existen métodos heurísticos de diversa naturaleza, sin embargo, estos pueden clasificarse en dos categorías:

Heurísticas constructivas: este tipo de heurísticas se encargan de construir una solución factible paso a paso, considerando el costo de la solución. Usualmente son métodos deterministas y no cuentan con fases de mejora. Dentro del estudio del UCTP este tipo de heurísticas suele usarse en la fase de construcción de las soluciones iniciales (Abuhamdah *et al*., 2014; Wahid y Hussin, 2016).

Heurísticas de búsqueda local: estas heurísticas parten de una solución factible y a partir de ella se hacen cambios de forma sistemática para mejorarla. Los cambios que hace el algoritmo dependen de la estrategia utilizada, múltiples estrategias de búsqueda local han sido utilizados en la literatura. Algunas de ellas son: first improvement, examina la vecindad y selecciona el primer movimiento que produce una mejora en la solución actual; best improvement, examina la vecindad

y todos los posibles movimientos seleccionando el mejor movimiento de todos los posibles; y randomized; en la que se selecciona aleatoriamente soluciones en la vecindad (Ortiz, 2010).

1.3.5.2.2 *Metaheurísticas:* Son algoritmos aproximados de optimización y búsqueda de alto nivel que guían una o varias heurísticas subordinadas combinando de forma inteligente distintos conceptos para explorar y explotar adecuadamente el espacio de búsqueda (Herrera, 2009).

Los métodos metaheurísticos aplicados a los problemas de horarios en general, según Lewis (2007) pueden clasificarse en tres grupos:

Algoritmos de optimización de una etapa: donde las restricciones duras y restricciones suaves son satisfechas simultáneamente.

Algoritmos de optimización de dos etapas: donde la satisfacción de las restricciones suaves solo se considera una vez que un horario factible haya sido encontrado.

Algoritmos que permiten relajación: las violaciones de las restricciones duras no se consideran al inicio del problema al relajar alguna característica del problema. Luego se intentan satisfacer las restricciones suaves a la vez que se van eliminando estas relajaciones.

Los algoritmos metaheurísticos según la cantidad de soluciones que manejen pueden clasificarse en dos tipos: metaheurísticas basadas en una solución única o basadas en poblaciones; dentro del primer grupo se encuentran los algoritmos de Búsqueda Tabú, Recocido Simulado, Gran Diluvio, VNS, GRASP, entre otros; en el segundo grupo, algunos algoritmos documentados son el Algoritmo Genético, la Optimización de Colonia de Hormigas, el algoritmo Artificial de Colonia de Abejas y el algoritmo de Búsqueda Armónica.

A continuación, se presentan algunos métodos metaheurísticos utilizados para resolver el UCTP.

- **Algoritmo genético (GA)**

Los algoritmos genéticos fueron popularizados por Holland en 1975. Esta metodología emplea operadores conocidos como operadores genéticos (como la selección, el cruce y la mutación) que manipulan soluciones individuales (denominadas cromosomas) en una población durante varias generaciones para mejorar la función objetivo (a menudo llamada la aptitud).

Un cromosoma se representa como una cadena de longitud fija donde cada posición se denomina gen y contiene una unidad de información de solución. Al usar un operador de selección, se escogen las mejores soluciones para convertirse en padres. El operador de cruce se usa para crear uno o más descendientes de los dos padres existentes. Algunas de las operaciones de cruce comunes son el cruce de un punto, el cruce de dos puntos, el cruce de ciclo y el cruce uniforme. Se deben considerar varios parámetros al aplicar un algoritmo genético a un problema dado, como el tamaño de la población, la tasa de cruce, la tasa de mutación y el número de generaciones (Salwani Abdullah, 2006).

- **Búsqueda tabú (TS)**

Esta metaheurística fue introducida por Fred Glover en 1986, es una técnica heurística global que trata de evitar caer en el óptimo local creando una lista especial llamada tabú. Cualquier solución que se haya seleccionado recientemente se coloca en una lista tabú para que se convierta en 'tabú'

durante un corto período de tiempo, dependiendo de la longitud de la lista. Esto minimiza la posibilidad de que se formen ciclos en la misma solución y, por lo tanto, crea más oportunidades de mejora al pasar a áreas no exploradas del espacio de búsqueda (Mushi, 2006).

- **Recocido simulado (SA)**

Este método es basado en la analogía del simulado de recocido de los metales. El término de recocido se refiere a un proceso físico en el que un sólido es calentado mediante temperaturas altas para que este pase a una fase liquida y luego sea enfriado lentamente mediante la disminución de la temperatura. De esta manera se dice que las partículas enfriadas poseen menos energía (Angeles, 2015).

Según (Tarawneh y Ayob, 2013) el recocido simulado es uno de los primeros algoritmos metaheurísticos que tienen por estrategia evitar los mínimos locales, permitiendo que soluciones de mala calidad sean aceptadas (con alguna probabilidad). SA siempre acepta soluciones vecinas con buena calidad; y probablemente acepta soluciones de peor calidad utilizando un criterio de probabilidad de aceptación $P(S_i)$.

$$P(S_i) = e^{-\Delta f / T_i}$$

Esta función de probabilidad controla la aceptación de una solución vecina s^* de calidad $f(s^*)$, usando la calidad de la solución actual $f(s)$ y la temperatura actual T_i, donde $\Delta f = f(s^*) - f(s)$. La temperatura actual T_i es reducida mediante un esquema de enfriamiento con un ratio de enfriamiento α para cada iteración. SA se detiene cuando una T_{min} es alcanzada.

- **Algoritmo de gran diluvio (GD)**

El algoritmo del gran diluvio es un enfoque metaheurístico propuesto por Dueck en 1993 y está inspirado en el comportamiento que podría surgir cuando alguien busca un terreno más alto para evitar el aumento del nivel del agua durante una lluvia constante.

Para un problema de maximización, el algoritmo busca encontrar el punto más alto en una cierta superficie con colinas, valles y mesetas (espacio de búsqueda). Luego, comienza a llover constantemente y el algoritmo camina (explora el vecindario) pero nunca da un paso hacia el creciente nivel del agua. A medida que continúa lloviendo, el algoritmo puede explorar el terreno más alto y más bajo (mejorando y no mejorando posiciones) pero se empuja continuamente a un punto alto (con suerte cercano al óptimo) hasta que finalmente no puede escapar del nivel de agua ascendente y se detiene. El nivel de agua inicial se establece en un valor por debajo de la idoneidad de la solución inicial y luego se incrementa de forma lineal a medida que avanza la búsqueda (Obit y Landa-silva, 2010).

- **Variable neighborhood search (VNS)**

Esta metaheurística fue VNS fue presentada por Mladenović y Hansen (1997). Se basa en la estrategia de utilizar más de una estructura de vecindario y cambiarlos sistemáticamente durante la búsqueda local. Esto ayuda al VNS a explorar una variedad de posibilidades y saltar a una nueva solución si es necesario. Otras técnicas en la literatura como el recocido simulado y la búsqueda tabú generalmente utilizan una única estructura de vecindario durante la búsqueda y generalmente se enfocan más en los parámetros que afectan la aceptación de los movimientos que en la estructura del vecindario (Abdullah, 2006, p.34).

- **Greedy randomized adaptative search procedures (GRASP)**

Propuesto por Feo y Resende en 1994, es un algoritmo metaheurístico que consta de una técnica de muestreo aleatorio en el cual por cada iteración proporciona una solución al problema. Existen dos fases por cada iteración del GRASP: fase de construcción y fase de mejora. En la primera se construye una solución inicial a partir de una función voraz adaptiva aleatoria y la segunda aplica un procedimiento de búsqueda local sobre la solución inicial con la esperanza de encontrar una solución mejor (Angeles, 2015).

- **Optimización de colonia de hormigas (ACO)**

La optimización de colonia de hormigas es una metaheurística basada en la población propuesta por Dorigo *et al*. (1996). La idea básica detrás de esta técnica de optimización se basa en la observación de que las hormigas encuentran su camino hacia una fuente de alimento y regresan a su nido. A medida que avanzan, se deposita un rastro de sustancia química (llamada feromona) que ayudará a otras hormigas a encontrar la misma fuente de alimento (Abdullah, 2006, p.25).

- **Colonia artificial de abejas (ABC)**

Es un algoritmo metaheurístico basado en la naturaleza propuesto por Karaboga en 2005 para optimizar los problemas numéricos. Fue motivado por el comportamiento de búsqueda inteligente de las abejas melíferas. El modelo comprende tres fundamentos vitales: abejas forrajeras empleadas y desempleadas, fuentes de alimentos y distancia a la colonia. las abejas forrajeras empleadas y desempleadas buscan fuentes de alimentos ricos, en las cercanías de la colmena (Bolaji, Khader, Al-Betar, y Awadallah, 2011).

- **Algoritmo de búsqueda armónica (HSA)**

El algoritmo de búsqueda de armonía (HSA) es un algoritmo metaheurístico basado en poblaciones desarrollado por Geem *et al*. en 2001. Este simula el proceso de improvisación musical, donde un grupo de músicos improvisa los tonos de sus instrumentos musicales, práctica tras práctica, buscando una armonía agradable determinada por un estándar de audio-estético. Esto se puede traducir en el proceso de optimización de la siguiente manera: a un conjunto de variables de decisión se le asignan valores, iteración por iteración, buscando una solución "lo suficientemente buena", evaluada mediante una función objetivo (Azmi y Tajudin, 2012).

- **Métodos híbridos**

El objetivo del enfoque híbrido es tomar la mejor idea de un enfoque e incorporarla con otra idea buena (o mejor) de otro enfoque, es decir, combina varios algoritmos heurísticos o metaheurísticos con el objetivo de mejorar el desempeño de los algoritmos aprovechando las ventajas que tiene cada enfoque. Cabe mencionar que muchos de los algoritmos ya descritos presentan métodos híbridos (en mayor o menor medida) que han sido aplicados al problema de programación de cursos universitarios (Salwani Abdullah, 2006).

1.3.6 **Híbrido de algoritmo genético y búsqueda tabú – HGATS.** Los autores Yang y Jat (2011) proponen un algoritmo de dos fases que combina el GSGA propuesto por ellos mismos en el año 2009 con una técnica de búsqueda local y una heurística TS para el problema de programación de cursos basados en la inscripción (PECTP), este nuevo algoritmo se denota HGATS.

El problema considerado por los autores considera las siguientes restricciones duras:

D1. Ningún estudiante puede atender más de un evento al mismo tiempo.

D2. El salón debe ser lo suficientemente grande para atender a todos los estudiantes y satisfacer todos los requisitos exigidos por el evento.

D3. Un salón solo puede albergar un evento a la vez.

D4. Los eventos deben ser asignados solamente en las franjas que están predefinidas como disponibles para esos eventos.

D5. Donde se especifique, los eventos deben ser programados en un orden correcto.

Además, la ocurrencia de alguna de las siguientes situaciones es penalizada como una restricción suave:

S1. Un estudiante tiene una clase en el último periodo del día.

S2. Un estudiante tiene más de dos clases en fila.

S3. Un estudiante tiene solamente una clase durante un día.

1.3.6.1 ***Formulación del problema.*** La formulación realizada por los autores del algoritmo, se planteó de la siguiente manera:

El problema de programación de cursos universitarios PECTP consiste en una serie de eventos n que deben ser programados en un conjunto de 45 franjas (9 periodos durante 5 días de la semana), un conjunto de m salones disponibles en los que los eventos pueden tomar lugar, un conjunto k de estudiantes quienes atienden los eventos, un conjunto l de prestaciones que son satisfechas por los salones y requeridas por los eventos.

Además de los conjuntos establecidos anteriormente, se requiere definir una serie de relaciones entre estos, para ello se vale de las siguientes matrices:

- Matriz estudiante-evento: esta matriz muestra qué eventos son atendidos por cuales estudiantes, cada celda de dicha matriz tiene el valor de 1 si un estudiante debe atender un evento y, 0 en caso contrario.
- Matriz evento-conflicto: indica que eventos pueden ser programados en la misma franja y cuáles no.
- Matriz salón-prestación: presenta los atributos que cada son posee, cada una de celdas de la matriz toma el valor de 1 si un salón cuenta con un atributo y 0 sino.
- Matriz evento-prestación: presenta los atributos requeridos por cada evento, tomando el valor de 1 si el evento requiere de cierto atributo.
- Matriz evento-salón: esta matriz enlista los posibles salones en los que cada evento puede ser asignado.
- Matriz evento-disponibilidad: toma el valor de 1 si un evento debería tomar lugar en cierta franja y 0 en caso contrario.

- Matriz evento-preferencia: en esta matriz se considera la restricción de precedencia entre eventos, en este caso, una celda puede tomar el valor de 1 si un evento debe ser programado antes que otro, o -1 si debe tomar lugar después de dicho evento. Si no hay restricción de precedencia entre el par de eventos el valor de la celda es 0.

Adicionalmente, también se plantean un conjunto de vectores que contienen la siguiente información:

- Vectores EE: cada vector EE_i es una lista de eventos que deben ser programados en un horario después de un evento en específico. Esta información ayuda a satisfacer la restricción D5 durante la ejecución del algoritmo.
- Vectores ET: cada vector ET_i es una lista de posibles franjas en donde un evento puede ser programado.
- Vector E'_e: este vector contiene un conjunto de eventos que no tiene ninguna restricción de tiempo.
- Vector T'_s: Contiene una lista de franjas que no tienen restricciones para ningún evento.

La solución es representada usando una lista ordenada de pares salón-franja (r_i, t_i), donde el índice de cada par es el número de identificación de un evento.

Tabla 2.
Representación de solución al problema PECTP usada por Yang y Jat.

r	**t**
2	4
3	10
1	12
…	…
2	7

Por ejemplo, para los pares representados en la tabla anterior, se observa que el salón 2 en la franja 4 fue asignado al evento 1. De forma similar, el evento 2 fue asignado al salón 3 en la franja 10.

En cuanto a la meta del PECTP el objetivo es minimizar las violaciones de las restricciones duras y suaves. Para ello, se define una función objetivo $f(s)$ para el horario s, como la suma del número de restricciones duras $\#hcv$ y restricciones suaves $\#scv$.

$$f(s) = \#hcv * C + \#svc$$

Donde C es una constante, cuyo valor es mayor que el máximo número posible de restricciones suaves.

1.3.6.2 ***Descripción del algoritmo.*** En la primera fase, el GSGA que usa operadores genéticos, una estrategia de búsqueda guiada y dos técnicas de búsqueda local, se usa para desarrollar una población de soluciones candidatas hacia soluciones cada vez mejores, idealmente para encontrar la solución óptima. Solamente si las soluciones encontradas durante la primera fase son factibles, la segunda fase es ejecutada, la cual usa una heurística de Búsqueda Tabú para mejorar la solución factible hacia la solución óptima.

La figura 4 presenta el pseudocódigo del enfoque híbrido HGATS donde se presenta de forma general el funcionamiento del algoritmo.

Algorithm 1 Proposed Hybrid Approach—HGATS

1: **input**: A problem instance **I**
2: set the generation counter $g := 0$
3: **for** $i := 1$ to population size **do**
4: $s_i \leftarrow InitializeIndividual(i)$
5: $s_i \leftarrow$ solution s_i after applying LS operator 1 (LS1)
6: $s_i \leftarrow$ solution s_i after applying LS operator 2 (LS2)
7: **end for**
8: **while** the termination condition is not reached **do**
9: **if** $(g \bmod \tau) == 0$ **then**
10: apply *ConstructMEM*() to construct *MEM*
11: **end if**
12: $s \leftarrow$ child solution by applying *GuidedSearchByMEM*() or *Crossover*() with a probability γ
13: $s \leftarrow$ child solution after mutation with a probability P_m
14: $s \leftarrow$ child solution after applying LS operator 1 (LS1)
15: $s \leftarrow$ child solution after applying LS operator 2 (LS2)
16: replace the worst individual of the population by s
17: $g := g + 1$
18: **end while**
19: **if** s is an optimal solution **then**
20: go to line 24
21: **else**
22: $s \leftarrow$ Apply *TabuHeuristic*() on the best solution obtained in the first phase
23: **end if**
24: **output**: The best solution s_{best} achieved for the problem instance **I**

Figura 3. Algoritmo HGATS. Adaptado de Naseem y Shengxiang (2011). Recuperado de: *https://doi.org/10.1007/s10951-010-0202-0*

GSGA se inicia a partir de una población inicial de individuos generados al azar, donde los eventos son asignados a salas y franjas para cada solución en función de la propiedad de cada evento. Con el objetivo de generar buenas soluciones iniciales, dos métodos de búsqueda local son aplicados a cada individuo de la población inicial, para ello, se hace uso de seis estructuras de vecindarios que en una primera instancia mueven eventos entre espacios de tiempo para luego aplicar un algoritmo de coincidencia para asignar los salones para cada evento

Después de la inicialización de la población, se construye una estructura de datos (denominada MEM), que selecciona buenos individuos de la población y almacena una lista de pares de salones y franjas (r, t) para los eventos que no tienen ninguna penalidad, es decir, no hay violaciones de restricciones duras y suaves para estos eventos. Después de eso, el MEM se puede utilizar para guiar la generación de descendientes de las siguientes generaciones. La estructura de datos MEM se reconstruye regularmente cada τ generaciones. En cada generación de GSGA, un hijo es generado mediante el uso de MEM o mediante la aplicación del operador de cruce, dependiendo de una probabilidad γ. Luego, el hijo se somete a la operación de mutación seguida de los métodos LS para una mejoría potencial. Finalmente, el peor miembro de la población se reemplaza con el individuo recién generado. El proceso de iteración continúa hasta que se alcanza una condición de terminación. Luego, si la mejor solución obtenida de la fase inicial es factible, la fase dos es ejecutada aplicando la heurística TS; si la solución no es factible, primero se eliminan todos los eventos que involucran violaciones de restricciones duras y se reconsideran solo si satisfacen todas las restricciones duras durante la búsqueda de vecindario.

1.3.6.3 ***Inicialización de la población.*** Cada individuo de la población inicial es creado por el algoritmo descrito en la figura 4. Los eventos están divididos en dos clases, los que no tienen restricciones en franjas particulares $\boldsymbol{E'_e}$ y los que sí. Si un evento no tiene restricciones en las franjas se le asignará una franja aleatoria y un salón disponible aleatorio; dado el otro caso, el evento será asignado en una franja aleatoria dentro de sus franjas disponibles ***ET*** y aleatoriamente le será asignado un salón disponible.

Algorithm 2 *InitializeIndividual*(i)

1: **input**: The index i of individual I_i
2: **for** each event e_j of I_i **do**
3: **if** event $e_j \in E'_e$ **then**
4: assign a random time slot from T'_s to e_j
5: assign a random room from a list of suitable rooms
6: **else**
7: assign a random time slot from ET_j to e_j
8: assign a random room from a list of suitable rooms
9: **end if**
10: **end for**
11: **output**: The generated individual I_i

Figura 4.Creación de un individuo. Adaptado de Naseem y Shengxiang (2011). Recuperado de: *https://doi.org/10.1007/s10951-010-0202-0*

De esta forma, el individuo generado satisface las restricciones duras D2 y D4. Sin embargo, aún no se garantiza la factibilidad y hace necesaria la aplicación de dos esquemas de búsqueda local (LS1 y LS2) que intentan hacerlo factible.

1.3.6.4 ***Métodos de búsqueda local.*** Los métodos de búsqueda local son aplicados en dos ocasiones dentro del algoritmo (cada vez que se genera un individuo y cuando se crea un descendiente), estos están basados en seis estructuras de vecindarios denotadas como N1, N2, N3, N4, N5 y N6 respectivamente. Cada una de ellas es descrita a continuación:

N1: la vecindad definida por un operador que mueve un evento de una franja a otra diferente.

N2: la vecindad definida por un operador que intercambia las franjas de dos eventos.

N3: la vecindad definida por un operador que permuta tres diferentes franjas de tres eventos en una de las otras dos posibles formas que existen para permutar tres eventos.

N4: la vecindad definida por un operador que toma dos eventos aleatorios del conjunto E_e' y reemplaza sus franjas por otras franjas aleatorias.

N5: la vecindad definida por un operador que toma cada evento que debe ocurrir en un orden y trata de encontrar un lugar en el horario para asignarlos.

N6: la vecindad definida por un operador que toma un subconjunto de franjas entre todas las franjas ocupadas. De este subconjunto la peor franja es seleccionada (que contiene eventos que colectivamente tienen el mayor valor de penalización) es seleccionada y sus eventos son movidos a otra franja aleatoriamente seleccionada en el subconjunto.

1.3.6.5 ***Operador de búsqueda local 1 (LS1).*** La figura 5 describe el pseudocódigo del operador, LS1 trabaja sobre todos los eventos y se divide en dos pasos. En el primer paso, se verifican las violaciones de restricciones dura de cada evento (ignorando las restricciones suaves). Si hay restricciones duras violadas por un evento, LS1 intenta resolverlos aplicando movimientos en las estructuras de vecindario N1, N2, N3 y N4 ordenadamente hasta que la condición de parada es alcanzada (por ejemplo, encontrar una mejora o alcanzar un número máximo de pasos).

Algorithm 6 Local Search Operator 1 (LS1)

```
1: input: Individual I from the population
2: for i := 1 to n do
3:    if event e_i is infeasible then
4:       if there is untried move left then
5:          calculate the moves: first in N1, then in N2 if N1
            fails, then in N3 if N2 also fails, and finally in
            N4 if N3 also fails
6:          apply the matching algorithm to the time slots
            affected by the move and delta evaluate the re-
            sult.
7:          if moves reduce hard-constraint violations then
8:             make the moves and go to line 4
9:          end if
10:      end if
11:   end if
12: end for
13: if no any hard-constraint violations remain then
14:    for i := 1 to n do
15:       if event i has soft-constraint violations then
16:          if there is untried move left then
17:             calculate the moves: first in N1, then in N2
                if N1 fails, then in N3 if N2 also fails, and
                finally in N4 if N3 also fails
18:             apply the matching algorithm to the time slots
                affected by the move and delta evaluate the
                result
19:             if moves reduce soft-constraint violations
                then
20:                make the moves and go to line 16
21:             end if
22:          end if
23:       end if
24:    end for
25: end if
26: output: A possibly improved individual I
```

Figura 5. Pseudocódigo de Búsqueda local 1. Adaptado de Naseem y Shengxiang (2011). Recuperado de: *https://doi.org/10.1007/s10951-010-0202-0*

Después de cada movimiento, se aplica el algoritmo de coincidencia (matching algorithm) a las franjas afectadas por el movimiento para intentar resolver la perturbación en la asignación de salones (cumpliendo la restricción D3), también, se realiza una evaluación delta, la cual calcula las violaciones de restricciones duras y suaves antes y después del movimiento. Si todos los movimientos ya han sido probados, LS1 continua al siguiente evento. Si luego de aplicar todas las vecindades en cada evento todavía hay violaciones en las restricciones duras, LS se detendrá; de otra forma, pasará al paso dos. En el segundo paso, después de alcanzar una solución factible, un proceso similar al paso uno es desarrollado en cada evento para reducir la violación de las restricciones suaves sin violar las restricciones duras.

1.3.6.6 ***Operador de búsqueda local 2 (LS2).*** La figura 6 describe el segundo operador de búsqueda local, LS2 trabaja sobre un conjunto de eventos con N5 (correspondiente a las líneas 2-9) y N6 (correspondiente a las líneas 10-27).

La idea básica de LS2 es, en primera instancia, intentar ubicar un evento involucrado en la restricción de precedencia D5 en una franja antes de su correspondiente lista de eventos sucesores. Después de mover el evento a una nueva franja usando N1 y N2, el nuevo valor de penalización es calculado. Si el movimiento reduce la penalidad, entonces este es guardado; de otra forma, es eliminado.

Después de usar N5, LS2 aplica N6. En este vecindario, primero se selecciona aleatoriamente un porcentaje de franjas del total de franjas disponibles en el horario. Entonces, el valor de penalización de cada franja seleccionada es calculado, luego, la peor franja es seleccionada para

la búsqueda local. Después de tomar la peor franja, LS2 intenta un movimiento en la estructura de vecindad N1 para cada evento de la franja y comprueba el valor de penalización de cada evento antes y después de aplicar el movimiento. Si todos los movimientos en la franja, juntos, reducen las violaciones de restricciones duras y / o blandas, entonces se aplican todos los movimientos; de lo contrario, no se hacen los movimientos. Cuando termina LS2, se obtiene un individuo posiblemente mejorado y factible.

Algorithm 7 Local Search Operator 2 (LS2)

1: **input**: Individual **I** after LS1 is applied
2: **for** each event e_i of I in EE **do**
3: **for** each event e_j in EE_i **do**
4: try to place event e_j in the timetable after the time slot of e_i by calculating a move of e_i in the neighbourhood N1 and N2
5: apply the matching algorithm to the time slots affected by the move
6: compute the penalty of e_i and delta evaluate the result
7: apply the move if it reduces hard- or soft-constraint violations
8: **end for**
9: **end for**
10: S:=randomly pick a percentage of occupied time slots from T
11: **for** each time slot $t_i \in S$ **do**
12: **for** each event e_j in the time slot t_i **do**
13: calculate the penalty value of event e_j
14: **end for**
15: sum the total penalty value of events in the time slot t_i
16: **end for**
17: select the time slot w_t with the biggest penalty value from S
18: **for** each event e_i in w_t **do**
19: calculate a move of e_i in the neighbourhood N1
20: apply the matching algorithm to the time slots affected by the move
21: compute the penalty of e_i and delta evaluate the result
22: **end for**
23: **if** all the moves of events in w_t together reduce hard- or soft-constraint violations **then**
24: apply the moves
25: **else**
26: delete the moves
27: **end if**
28: **output**: A possibly improved individual **I**

Figura 6. Pseudocódigo de Búsqueda local 2. Adaptado de Naseem y Shengxiang (2011). Recuperado de: *https://doi.org/10.1007/s10951-010-0202-0*

1.3.6.7 ***Estructura de datos (MEM).*** Los autores proponen una estructura de datos MEM para proporcionar una mayor dirección de exploración y explotación en el espacio de búsqueda. La estructura de datos está conformada por un esquema de dos niveles; el primer nivel es una lista de eventos y el segundo nivel es una lista $\boldsymbol{l_i}$ de pares salón-franja correspondiente a cada evento $\boldsymbol{e_i}$ en el primer nivel.

La estructura de datos MEM es reconstruida cada τ generaciones, en la figura 7 muestra el bosquejo de la construcción de la MEM. Durante el proceso de reconstrucción, primero se seleccionan $\alpha \times N$ de los mejores individuos desde la población P para formar un conjunto Q, donde N denota el tamaño de la población. Luego, a cada individuo $I_j \in Q$, se le revisa la penalidad de cada evento $e_i \in E'_e$. Si el evento tiene cero penalidades, entonces, se almacena la información correspondiente a ese evento dentro de la MEM. De forma similar, se aplica este procedimiento para el siguiente individuo en Q hasta que se almacenen los mejores pares salón-franja para cada evento. Esta estructura de datos es usada para guiar la creación de descendientes durante las siguientes τ generaciones.

Algorithm 3 *ConstructMEM*()

```
1: input: The whole population P with the population size
   N
2: sort the population P according to the fitness of individ-
   uals
3: Q ← select the best α × N individuals in P
4: for each individual I_j in Q do
5:    for each event (e_i ∈ E'_e) in I_j do
6:       calculate the penalty value of event e_i from I_j
7:       if e_i is feasible (i.e., e_i has zero penalty) then
8:          add the room and time slot pair (r_i, t_i) assigned
            to e_i into the list l_i
9:       end if
10:   end for
11: end for
12: output: The data structure MEM
```

Figura 7. Construcción de la estructura de datos MEM. Adaptado de Naseem y Shengxiang (2011). Recuperado de: *https://doi.org/10.1007/s10951-010-0202.*

1.3.6.8 ***Operador de búsqueda guiada por MEM.*** En GSGA, un hijo es creado aplicando la búsqueda guiada por MEM o usando un operador de cruce con una probabilidad $\boldsymbol{\gamma}$. La figura 8 presenta de forma resumida el procedimiento para generar un hijo con el operador de búsqueda guiada.

Algorithm 4 *GuidedSearchByMEM*()

1: **input**: The *MEM* data structure
2: $E_s :=$ randomly select $\beta * |E'_e|$ events from E'_e
3: **for** each event e_i in E_s **do**
4: randomly select a room and time slot pair from the list l_i
5: assign the selected pair to event e_i for the child
6: **end for**
7: **for** each remaining event e_i not in E_s **do**
8: **if** $e_i \in ET$ **then**
9: assign a particular time slot and suitable room to e_i
10: **else**
11: assign a random time slot and room to e_i
12: **end if**
13: **end for**
14: **output**: A new child generated using *MEM*

Figura 8. Búsqueda guiada por MEM. Adaptado de Naseem y Shengxiang (2011). Recuperado de: *https://doi.org/10.1007/s10951-010-0202-0*

Si un hijo va a ser creado usando la MEM, primero se selecciona un subconjunto E_s de eventos aleatorios. Para cada evento en E_s se escoge un par salón-franja aleatorio de la lista l_i y se asigna el par seleccionado al evento e_i del hijo. Si hay algún evento en E_s que no tenga pares asignados en la MEM, aleatoriamente se le asigna un salón y una franja adecuada para el evento. Los eventos no incluidos en E_s son asignados a franjas y salones de acuerdo a los requerimientos particulares del hijo.

1.3.6.9 ***Operador de cruce.*** Si un hijo va a ser generado por el operador de cruce, primero se seleccionan dos individuos de la población actual como padres usando el criterio de Tournament Selection con tamaño igual a 2. Luego, un hijo es generado como sigue: para cada evento, se selecciona el padre con menor valor de penalización correspondiente a ese evento, y se asigna el par salón-franja correspondiente al evento del hijo. La figura 9 resume este procedimiento.

Algorithm 5 *Crossover*()

1: **input**: The current population
2: Select parents $P1$ and $P2$ by the tournament selection
3: **for** each event e_i of the child Ch **do**
4: **if** penalty value of e_i of $P1$ $\prec$ penalty value of e_i of $P2$ **then**
5: e_i of $Ch \leftarrow$ the time slot and room allocated to e_i of $P1$
6: **else**
7: e_i of $Ch \leftarrow$ the time slot and room allocated to e_i of $P2$
8: **end if**
9: **end for**
10: **output**: A new child generated using crossover

Figura 9. Operador de cruce. Adaptado de Naseem y Shengxiang (2011). Recuperado de: *https://doi.org/10.1007/s10951-010-0202-0*

1.3.6.10 ***Operador de mutación.*** Después de que se genera un niño utilizando MEM o cruce, se usa un operador de mutación con una probabilidad $\boldsymbol{P_m}$. El operador de la mutación primero selecciona aleatoriamente una de cuatro estructuras vecinales N1, N2, N3 y N4, y luego hace un movimiento dentro de la estructura del barrio seleccionado.

1.3.6.11 ***Heurística de Búsqueda Tabú.*** La heurística TS utilizada en HGATS es similar al esquema TS descrito en Rossi-Doria et al. (2002). Inicialmente, se verifica la mejor solución obtenida de la primera fase. Si es óptima, la Fase II no se ejecuta. Si es una solución factible, entonces se mejora la solución aplicando la heurística TS; si la solución no es factible, primero se eliminan todos los eventos que involucran violaciones de restricciones duras y se reconsideran solo si satisfacen todas las restricciones duras durante la búsqueda de vecindario.

En esta heurística, se aplica N1, N2 y N4 como estructuras vecinales para mover una solución. El movimiento de una solución se define moviendo un evento aleatorio de la solución usando N1, intercambiando dos eventos aleatorios de la solución usando N2, o intercambiando dos eventos específicos de la solución por intervalos de tiempo usando N4, de manera ordenada. La figura 10 resume la fase dos del algoritmo HGATS.

Algorithm 8 *TabuHeuristic*()—Phase II of HGATS

1: **input**: The best solution s_{best} from Phase I (GSGA)
2: $s \leftarrow s_{best}$
3: **if** s is not feasible **then**
4: remove all events that involve hard-constraint violations
5: **end if**
6: $TL \leftarrow \emptyset$
7: **while** the termination condition is not reached **do**
8: **for** $i := 0$ to 10% of the neighbours **do**
9: $s_i \leftarrow s$ after the ith move
10: compute the objective value $f(s_i)$
11: **end for**
12: **if** $\exists s_j | f(s_j) < f(s)$ and $f(s_j) \leq f(s_i)\ \forall i$ **then**
13: $s \leftarrow s_j$
14: $TL \leftarrow TL \cup E_i$ where E_i is the set of events moved to get s_j
15: **else**
16: $s \leftarrow$ the best non-tabu moves among all s_i
17: $TL \leftarrow TL \cup E_b$ where E_b is the set of events moved by the best non-tabu move
18: **end if**
19: $s_{best} \leftarrow$ the best solution so far
20: **end while**
21: **output**: The optimised solution s_{best}

Figura 10. Heurística de búsqueda tabú. Adaptado de Naseem y Shengxiang (2011). Recuperado de: *https://doi.org/10.1007/s10951-010-0202-0*

Un movimiento es tabú si al menos uno de los eventos involucrados se ha movido en los últimos l pasos, donde l es la longitud de la lista tabú TL. La longitud de la lista tabú se establece como el número de eventos dividido por una constante K (K = 100 como se describe en Rossi-Doria *et al.* (2002)). Para disminuir la probabilidad de generar ciclos de movimientos y mejorar la exploración, se aplica un conjunto de vecindad variable, como se sugiere en Rossi-Doria et al. (2002), donde cada movimiento usa el vecindario N1, N2 o N4 con una probabilidad de 0.1. Para explorar el espacio de búsqueda de manera más eficiente, se acepta un movimiento tabú si supera la mejor solución hasta el momento. La heurística TS continúa hasta que se alcanza un límite de tiempo o se obtiene una solución óptima.

1.3.7 **Metodología.**

1.3.7.1 ***Revisión de literatura.*** La tabla 3 resume el procedimiento establecido para llevar a cabo la revisión de literatura

Tabla 3.
Metodología de revisión de literatura

Actividades	**Acción**
Paso 1: Definición del tema de investigación.	Se especifica el problema que se desea investigar
Paso 2: Identificación de la literatura a revisar	Establecer bases de datos y fuentes de información relevantes Definición de palabras claves Definición de la ecuación de búsqueda
Paso 3: Análisis de literatura	Revisión rápida de los artículos Agrupación de los artículos por categorías Establecimiento de criterios de exclusión e inclusión. Selección de estadísticas a incluir en la revisión de literatura Identificación de tendencias y patrones en los artículos
Paso 4: Resumen de literatura	Resumir organizadamente los artículos de tal forma que permitan identificar de manera ágil los aportes de interés
Paso 5: Síntesis y escritura de la revisión	Organizar los artículos que harán parte de la revisión y redactar la revisión de literatura

Nota. Adaptación de la metodología para escritura de revisiones de literatura. Adaptado de "Guidelines for writing a literature review". University of Minnesota Duluth. Duluth; 2014.

1.3.7.2 ***Descripción del problema.*** Se describe de forma general el problema de programación de cursos universitarios de la Escuela de Estudios Industriales Empresariales de la Universidad Industrial de Santander para el programa de Ingeniería Industrial (caso EEIE). Finalmente, se propone un modelo de programación lineal que contiene las restricciones y características consideradas en EEIE.

1.3.7.3 ***Método de solución.*** La técnica seleccionada corresponde a la metaheurística hibrida HGATS propuesta por Jat y Yang (2011) que combina un Algoritmo Genético con Búsqueda Guiada y el algoritmo de Búsqueda Tabú. En esta etapa se hace la descripción general de la técnica, se realiza la modificación del algoritmo para adaptarlo al problema en cuestión y se describe el seudocódigo del algoritmo.

1.3.7.4 ***Experimentación.*** Se organizan las instancias a ser tratadas por la metaheurística. En esta fase, también, se realiza el ajuste de parámetros de la metaheurística, y se obtiene la solución de cada una de las instancias.

1.3.7.5 ***Evaluación de resultados.*** En esta etapa se evalua el desempeño de la metaheurística en términos de tiempo y calidad de la solución respecto al horario elaborado por el Coordinador Académico en el semestre 2018-1 y 2018-2.

1.3.7.6 ***Elaboración del artículo de carácter publicable.*** Respecto a la elaboración del artículo de carácter publicable, en esta etapa se lleva a cabo la síntesis de la información relevante obtenida en el proyecto de investigación y se adapta el documento a las normas de la revista: IJIE - International Journal of Industrial Engineering: Theory Applications and Practice.

2. Revisión de literatura

El apéndice A presenta el desarrollo de la metodología usada para llevar a cabo la revisión de literatura.

El problema de programación de cursos universitarios hace parte de un amplio grupo de problemas de programación, comúnmente conocido como “Timetabling”, este tipo de problemas pueden definirse de forma general como la asignación de un conjunto de eventos a un espacio y tiempo cumpliendo una serie de restricciones que involucran un conjunto de recursos escasos (Obit, 2010).

Los problemas de programación de horarios tienen aplicaciones en diversos campos de estudio como educación, transporte, deportes, salud, etc. Particularmente, en el caso de la educación, Schaerf y Di Gaspero (2001) dividen el problema en tres tipos: programación de horarios en colegios, programación de exámenes y programación de cursos universitarios; este último es el que ocupa al presente trabajo, también cabe mencionar que en documentos en español suele

referirse a este problema como generación de horarios universitarios, calendarización de materias, asignación de horarios universitarios o programación de asignaturas.

El estudio formal de este problema empezó con Gotlieb en 1963, desde entonces profesionales e investigadores han venido proponiendo diferentes métodos y procedimientos para solucionar el problema del UCTP. Welsh & Powell en 1967 resolvieron el primer problema de horarios utilizando el problema de coloración del gráfico (graph coloring problem) en el que reducía el problema a una serie de vértices (cursos) y arcos (conflictos) donde los periodos estaban representados por colores; sin embargo, este enfoque no lograba abordar instancias grandes del problema.

En 2002, se llevó a cabo la primera edición de la Competencia Internacional de Horarios (ITC-2002) organizada por Metaheuristic Networks, en este evento se propuso un problema tipo con tres restricciones duras y tres restricciones blandas junto a un conjunto de 20 instancias con el objetivo que investigadores y profesionales aplicaran diferentes métodos para abordar el problema. En años posteriores este problema y sus instancias fueron usadas como referencia para que más investigadores probaran la efectividad de sus propias técnicas de solución.

En 2003, Rossi-doria *et al*. llevaron a cabo la comparación de cinco metaheurísticas para comprobar su desempeño en la solución de un problema de programación de cursos universitarios, los algoritmos evaluados fueron el Algoritmo Evolutivo, Optimización de Colonia de Hormigas, Búsqueda Local Iterada, Recocido Simulado y Búsqueda Tabú, con el fin de evitar sesgos, las implementaciones de todos los algoritmos utilizaron una representación de solución común y una

misma estructura de vecindades. Finalmente, concluyeron que no era posible establecer que técnica era la mejor.

Avella y Vasil'ev (2005) publicaron lo que sería la primera formulación de enteros que considera explícitamente una amplia gama de requerimientos, lo cual lo hacía un problema computacionalmente desafiante. Los autores trabajaron el problema de programación de cursos universitarios de la facultad de ingeniería de la Universidad del Sannio en Italia como un problema de programación lineal en el que implementaron dos familias de planos cortantes como métodos de relajación, desigualdades Clique y Lifted Odd-Hole, y desarrollaron un algoritmo Branch-and-Cut capaz de encontrar soluciones optimas de un conjunto de instancias del mundo real.

Posteriormente, Schimmelpfeng y Helber (2007) definen un modelo de programación entera mixta multi-objetivo para abordar el problema UCTP de la Escuela de Economía y Gerencia de la Universidad de Hannover, en este modelo, los requerimientos que generalmente serian tomados como restricciones duras, fueron incorporados en la función objetivo y penalizados en medida a la importancia de la restricción con el fin de garantizar una solución "optima" con el menor número de conflictos.

En el año 2007, se llevó a cabo la segunda edición de la Competencia Internacional de Horarios (ITC-2007) agregando complejidad al planteamiento original, en esta competición el problema de programación de cursos universitarios se separó en tres problemas diferentes; programación de cursos basados en el plan de estudios (CB-CTT, Curriculum Based Course Timetabling), programación de cursos basados en la inscripción (PE-CTT, Post Enrolment-based Course

Timetabling) y programación de exámenes. La principal diferencia entre los dos primeros radica en que en el CB-CTT, los cursos se programan de acuerdo con los planes de estudio publicados por la universidad, mientras que en el PE-CTT, los cursos se programan de acuerdo con los datos de inscripción de los estudiantes. Para cada problema se definieron diferentes instancias de referencia tal como se hizo en la primera competición.

Abdullah, Burke, y Mccollum (2007) trabajaron un método llamado Algoritmo de Mejoramiento Iterativo Aleatorizado, el algoritmo inicia una solución factible generada por una heurística constructiva que luego pasa a un proceso de iteración en el que intervienen 11 estructuras de vecindarios, en este método una mala solución puede ser aceptada con cierta probabilidad. El algoritmo fue evaluado con instancias propuestas por 'Metaheuristic Networks' y demostró ser eficiente en instancias pequeñas, sin embargo, en instancias medianas el tiempo computacional aumentó significativamente y para la instancia más grande no le fue posible encontrar una solución factible.

Al-betar & Khader (2010) proponen un Algoritmo de Búsqueda Armónica (MHSA), en esta técnica la Memoria Armónica se llena con soluciones factibles generadas aleatoriamente usando los algoritmos Backtracking y Multiswap. Durante el proceso de Improvisación se usa el Smallest Position Algorithm para asignar los cursos según el número de posibles posiciones factibles y seguidamente aplica el Ajuste de Tono con tres posibles movimientos, luego, si se requiere se emplea un proceso de reparación, y, finalmente, se remplaza la peor armonía con la nueva solución generada. Esta metaheurística fue probada con 11 instancias de 'Socha benchmark' logrando las mejores soluciones registradas a esa fecha en dos de las instancias más complejas. En ese mismo

año, Al-betar, Khader, y Liao modificaron su algoritmo introduciendo cinco nuevos movimientos en el ajuste de tono (HSA-MPAR) , sin embargo, este no superó el desempeño de la metaheurística usada inicialmente. Posteriormente, en 2012, Al-Betar, Khader y Zaman crean un nuevo algoritmo hibrido de búsqueda armónica (HHSA), este nuevo algoritmo se basó en el MHSA, propuesto por los autores en su trabajo previo, introduciendo un optimizador Hill Climber que es aplicado cada vez que se genera una nueva armonía. Al momento de comparar el HHSA se encontró que este superaba a las anteriores propuestas hechas por los autores.

Abuhamdah y Ayob (2010) usan un método llamado Algoritmo de Descenso Adaptativo Aleatorizado (ARDA) basado en la estrategia LACH propuesta por Burke y Bykov (2008), este algoritmo a diferencia de LACH permite la aceptación de soluciones ligeramente peores a la solución actual o al promedio de soluciones en la lista de soluciones aceptables. La evaluación del algoritmo se hace sobre instancias de 'Socha benchmark' logrando soluciones que se equiparan en calidad a las mejores metaheurísticas disponibles en ese momento.

Obit y Landa-silva (2010) proponen una versión modificada del algoritmo de gran diluvio llamado Algoritmo de gran diluvio no lineal (NLGD), en esta propuesta, la tasa de disminución del nivel del agua es remplazada por una función no lineal que disminuye en cada iteración. El algoritmo aplica tres estructuras de vecindarios para generar una solución inicial factible. Además, estas mismas estructuras se usan aleatoriamente para guiar los movimientos en cada iteración. El algoritmo fue probado en 11 instancias de 'Socha benchmark' y 20 instancias del ITC-2002.

Yang y Jat (2011) usando como referencia las instancias de ‘Socha benchmark’ evalúan el efecto que se obtiene al incluir estrategias de Búsqueda Local y Búsqueda Guiada por Memoria al Algoritmo Genético básico; como resultado las siguientes metaheurísticas fueron propuestas: SSGA, GALS, GSGA, EGSGA; el SSGA es un algoritmo genético con una heurística de búsqueda local aplicada a cada individuo de la población, el GALS es un SSGA con una segunda estrategia de búsqueda local aplicada luego de la primera estrategia de búsqueda local, el GSGA es un SSGA con una estructura de memoria y una búsqueda guiada por dicha memoria, finalmente, el EGSGA es un GSGA con una nueva heurística de búsqueda local. En la experimentación, se encontró que el EGSGA obtenía mejores soluciones en el mismo tiempo de corrida. En ese mismo año, los autores proponen un algoritmo HGATS de dos fases basado en un GSGA hibridado con un algoritmo de Búsqueda tabú, la evaluación de la metaheurística se hace con instancias de referencia del problema PE-CTT de ITC-2007.

Abdullah, Shaker y Shaker (2011) establecen una estrategia Round Robin para seleccionar uno de tres algoritmos al momento de explorar el espacio de búsqueda, los algoritmos considerados son los de Escalada Simple (Hill Climbing), Gran Diluvio y Recocido Simulado. Este método es probado con dos problemas diferentes, el primero corresponde al planteado en ITC-2002 con instancias de Socha (2002) y el problema propuesto para CB-CTT de ITC-2007 con instancias de UD1.

Chavez, Pozos y Lengyel (2011) diseñaron e implementaron una heurística directa que usa una familia de mecanismos basados en clasificación de eventos llamada Sort Then Fix (STF), los autores ordenan los eventos más restringidos usando 11 criterios de clasificación con el fin de ir

asignando cada evento hasta completar el horario. En el trabajo, se diseñan 30 diferentes mecanismos STF con el fin de comparar la efectividad de los criterios usados. Finalmente, la heurística es probada en el problema estándar de ITC-2002 con 20 instancias de referencia y en el problema de PE-CTT con 24 instancias, el algoritmo demuestra ser eficiente en tiempo computacional, sin embargo, la calidad de las soluciones no se acerca a las mejores soluciones obtenidas con métodos metaheurísticos.

Kohshori y Abadeh (2012) trabajaron sobre el problema CB-CTT de ICT-2007 para comparar la efectividad de tres metaheurísticas híbridas basadas en el Algoritmo Genético (GA). En cada una de las nuevas metaheurísticas, los algoritmos de Búsqueda Local Iterada Aleatoria, Recocido Simulado y Búsqueda Tabú son introducidos luego del proceso de Mutación, obteniendo: el FGASI, que hibrida el GA con búsqueda local iterada aleatoria, el FGASA, el cual introduce la técnica de Recocido Simulado, y finalmente, el FGATS, que usa la Búsqueda Tabú en el proceso de mejora de los descendientes. En la experimentación se encontró qué, aunque no es posible elegir la mejor metaheurística, FGATS es capaz de entregar soluciones aceptables en un menor tiempo computacional.

Burke *et al.*, (2012) proponen un modelo de programación multiobjetivo para abordar el problema CB-CTT de la Competencia Internacional de Horarios de 2007, dada la complejidad del problema y el alto tiempo de ejecución, los autores aplican cinco cortes como métodos de relajación. Al evaluar el modelo, se logró igualar la mejor solución en dos instancias de referencia del Track 3 de ITC-2007 en menos de 15 minutos.

Lach y Lübbecke (2012) definen un modelo de programación lineal que opera en dos fases, en la primera fase se produce un horario factible que optimiza las restricciones de currículo compacto, asignar salones adecuados y la distribución de los cursos en diferentes días de la semana, mientras que, en la segunda fase se busca optimizar la estabilidad de salones para los cursos.

Abdullah, Turabieh, McCollum y McMullan (2012) estudian un método hibrido que combina las propiedades del algoritmo de Gran Diluvio (GD) con un Mecanismo de Tipo Electromagnético (EM), este mecanismo se basa en la propiedad física de atracción y repulsión de los puntos en camino a la solución óptima. En este híbrido el mecanismo de atracción-repulsión se encarga de actualizar el "nivel" en GD dentro del proceso de búsqueda. Este método trabaja sobre soluciones factibles generadas por una heurística llamada Largest Degree Heuristic, dos estructuras de vecindario y el algoritmo de Búsqueda Tabú. La experimentación se llevó a cabo sobre 11 instancias de Socha (2002) y el conjunto de instancias UD2 del problema CB-CTT de ITC-2007.

Chinnasri y Sureerattanan (2012) realizaron un estudio en el que se compara el desempeño de tres operadores de cruce en un Algoritmo Genético básico. Los operadores evaluados fueron el cruce parcialmente compatible (PMX), el cruce ordenado (OX) y el cruce cíclico (CX). La aplicación de la metaheurística se hace considerando el problema de Socha (2002) e incluyendo la disponibilidad de los profesores como una restricción suave adicional. Los autores concluyen que OX es el más eficiente al generar soluciones factibles, mientras que CX es un buen operador para encontrar soluciones óptimas. Además, encontraron que el proceso de mutación es más eficiente cuando pocos eventos intervienen en la iteración.

Tarawneh, Ayob y Ahmad (2013) abordan el problema CB-CTT de ITC-2007 usando una metaheurística hibrida del algoritmo de Recocido Simulado (SAM) que almacena soluciones en una memoria con el fin de escapar de los óptimos locales cuando la temperatura se ha hecho muy baja. El algoritmo funciona en dos fases, en la primera una heurística secuencial voraz y la Búsqueda de Descenso Empinado son usadas para generar una solución factible, mientras que, en la fase dos se minimizan las violaciones de las restricciones suaves, usando tres estructuras de vecindarios que crearán las soluciones que serán almacenadas.

Chen y Shih (2013) trabajan sobre un problema orientado a las universidades Taiwanesas, los autores evalúan el desempeño de cuatro algoritmos basados en la Optimización de Enjambre de Partículas (PSO). En primer lugar, se encuentra el PSO básico al que se le agrega un valor de peso inercial a la velocidad de la partícula; el segundo algoritmo es un PSO básico al que se le añade un factor de constricción sobre la velocidad, el factor social y el factor cognitivo de la partícula, este algoritmo se denota como Standard PSO (SPSO); al incluir un procedimiento de búsqueda local en PSO y SPSO, se crean las metaheurísticas hibridas PSOLS y SPSOLS. En la experimentación los autores confirman que la integración de mecanismos de búsqueda local mejora la calidad de las soluciones obtenidas, especialmente para el SPSOLS.

Badoni, Gupta y Mishra (2014) usan como referencia el problema del ITC-2002, los autores proponen un nuevo algoritmo hibrido de Algoritmo Genético con Búsqueda Local (NHA), en este método la instancia es tratada con una heurística de agrupación que se encarga de agrupar eventos que comparten estudiantes, luego el algoritmo GALS se encarga de generar las soluciones y explorar el espacio de búsqueda. La evaluación se hizo con dos tipos de instancias, en la primera

se comparó GALS y NHA en las 20 instancias del ITC-2002 y se encontró que NHA obtenía mejores soluciones en todos los casos, en la segunda evaluación se consideraron 12 instancias de Rossi-Doria *et al.* (2003), la comparación se hizo con otras 12 metaheurísticas siendo NHA la primera en encontrar una solución para la instancia hard02.

Wibowo (2014) propone una técnica metaheurística conocida como ACOA (Adapted Cuckoo Optimization Algorithm), la cual está derivada del algoritmo de Búsqueda Cuco (CS) y el Algoritmo de Optimización Cuco (COA), este algoritmo se inspira en el estilo de vida parasitaria de las crías de especies de aves conocidas como cucos, donde las aves se reproducen rápidamente depositando sus huevos en los nidos de otro ave huésped. La metaheurística es probada y comparada con el Algoritmo genético sobre cinco instancias de la Universidad Tecnológica de Malasia – UTM.

Abuhamdah *et al.* (2014) plantean un algoritmo de Búsqueda Local Basado en Poblaciones (PB-LS), en el que se busca aumentar la capacidad de intensificación de los algoritmos basados en poblaciones usando una población de soluciones en la búsqueda local, en este caso la heurística de búsqueda local usada fue MPCA-ARDA, propuesta por los autores en un trabajo previo, mientras que la fase de diversificación estaba a cargo de una adaptación del algoritmo GELS basado en los principios de la atracción gravitacional. En este trabajo, los autores deseaban probar qué tanto mejoraba MPCA-ARDA al incluirlo en el PB-LS, para ello, se evaluaron 11 instancias del ya conocido problema de Socha (2002).

Nagata y Ono (2014) proponen un algoritmo de dos fases; en la primera fase una solucion inicial factble es generada asignando eventos ordenados aleatoriamente a un horario vacío mientras se cumplen las restricciones duras; en la segunda fase, un algoritmo de Búsqueda Tabú con Vecindades Aleatorias Parciales conocido como RPNS (Random Partial Neighborhood Search) se encarga de mejorar la calidad de la solución, en esta técnica dos estructuras de vecindarios de tamaño variable se usan para intercambiar los eventos de cada candidato. En la experimentación, el RPSN demuestra ser altamente efectivo al evaluarse 5 instancias medianas y 1 instancia grande de Socha (2002).

Zheng, Wang, Liu y Zhang (2015) trabajan sobre el problema CB-CTT de ITC-2007 incluyendo una restricción suave que considera el tiempo de transporte para estudiantes que atienden eventos consecutivos y tienen que moverse entre edificios. Los autores proponen un algoritmo de Recocido Simulado que recuerda la mejor solución obtenida (ISAM), esta metaheurística opera en dos etapas; la primera genera una solución inicial, no necesariamente factible, siguiendo un procedimiento secuencial; la segunda etapa opera sobre la mejor solución usando cuatro tipos de movimientos en el proceso de búsqueda local. En el artículo se evalúa el desempeño del algoritmo bajo diferentes parámetros y como el peso de la nueva restricción afecta la calidad del resultado.

Abuhamdah (2015) diseñan un algoritmo de Criterio de Aceptación Adaptativo (AAC), este comienza con la generación de una solución inicial y explora iterativamente sus soluciones vecinas, buscando una mejor. La solución vecina se logra mediante la reestructuración de la solución actual utilizando algunas estructuras vecinales. Siendo AAC una metaheurística de búsqueda local, el autor propone un mecanismo similar al usado en ARDA para escapar de los

óptimos locales. El algoritmo es evaluado en 11 instancias de Socha (2002) y al ser comparada con otras técnicas de la literatura solo las iguala en pequeñas instancias.

Vermuyten, Lemmens, Marques y Beliën (2016) proponen un modelo de programación lineal para abordar el UCTP de la Facultad de Economía y Negocios de KU Leuven en Bélgica, en este problema se considera el desplazamiento entre edificios como un objetivo a minimizar. Dado el tamaño del problema, los autores proponen un modelo de dos etapas; en la primera fase, se busca un horario que cumpla con las restricciones duras del problema y minimice las violaciones en las preferencias de los profesores, mientras que la segunda fase usa el horario de la primera etapa como entrada con el objetivo de minimizar el flujo de estudiantes mediante la reasignación de eventos a los salones.

Teoh, Haron y Wibowo (2016) estudian un Algoritmo Genético con Búsqueda de Vecindarios (GANS) para resolver el problema de programación de horarios de una facultad en una universidad de Malasia. Este enfoque inicia por la aplicación del Algoritmo Genético para generar una familia de soluciones factibles, luego la mejor solución es seleccionada y pasa a un proceso de explotación por parte de tres estructuras de vecindario que guían la solución hacia el óptimo global.

Wahid y Hussin (2017) diseñan una metaheurística hibrida entre el algoritmo de Búsqueda Armónica y el algoritmo de Gran Diluvio para abordar el problema UUM CAS CB-CTT del ITC-2007, en esta técnica el algoritmo GD se incorpora en la fase de Consideración Aleatoria del HSA con el objetivo de balancear la capacidad de exploración e intensificación del algoritmo. En la

evaluación, se obtiene que la nueva metaheurística ofrece soluciones de mejor calidad que las obtenidas con un paquete de software.

Bucco y Bandeira (2017) proponen un algoritmo de programación lineal de dos etapas para abordar un UCTP del mundo real en una universidad de Brasil. En la primera etapa se asignan eventos a periodos minimizando el número de eventos asignados en el periodo con la mayor cantidad de eventos, de esta forma se reduce el número total de salones necesarios, es decir, si en un periodo se asignan 14 eventos, entonces se requerirán por lo menos 14 salones para poder atender esos eventos; luego la segunda fase usa el primer horario como parámetro de entrada para llevar a cabo la asignación de salones minimizando el número de unidades académicas que sean requeridas.

Como ya se mencionó diferentes algoritmos han sido propuestos y evaluados usando instancias de referencia de las competencias internacionales; sin embargo, es interesante estudiar el desempeño que estos métodos tendrán en instancias del mundo real. Blaz (2016) lleva a cabo la aplicación del algoritmo EGSGA propuesto por Yang y Jat para realizar un sistema de generación de horarios utilizando algoritmos genéticos en universidades peruanas.

3. Descripción del problema

La Escuela de Estudios Industriales y Empresariales (EEIE) de la Universidad Industrial de Santander, con el fin de llevar a cabo el cumplimiento de su función misional en Docencia,

semestralmente, realiza la Planeación de Matrícula para los programas Académicos de Pregrado y Posgrado ofrecidos por esta.

En el caso de programa de Pregrado de Ingeniería Industrial, el proceso de Planeación de Matrícula se detalla en el Apéndice B y abarca la programación de las asignaturas por parte del Coordinador Académico del programa, la solicitud de matrícula de las asignaturas en el sistema por parte de cada estudiante y el informe de oferta y demanda utilizado para ajustar la programación de asignaturas previamente hecha. El proceso de Planeación de Matrícula finaliza con la publicación vía web de los horarios para luego habilitar un periodo de tiempo en el cual los estudiantes pueden realizar inclusiones, cancelaciones y/o ajuste de asignaturas.

El presente proyecto se centra en el proceso de Planeación de Matrícula, específicamente, aborda la programación de asignaturas realizada por el Coordinador Académico en la Escuela de Estudios Industriales y Empresariales (EEIE) de la Universidad Industrial de Santander para el programa de pregrado de Ingeniería Industrial en su ciclo profesional.

3.1 Programación de asignaturas en EEIE

El caso de estudio en EEIE puede ser catalogado dentro del conjunto de problemas de UCTP conocido como CB-CTT (*Programación de cursos basados en el currículo*), esta rama se caracteriza por realizar la planeación de los horarios basado en el plan de estudio del programa académico y considerando una proyección de la cantidad de cursos que serán requeridos en el siguiente semestre.

3.1.1 **Definiciones.** La escuela de estudios industriales y empresariales de la Universidad industrial de Santander lleva a cabo la programación de los horarios del programa de Ingeniería Industrial considerando las asignaturas, profesores, salones y franjas horarias disponibles durante una semana para luego replicar esta semana a lo largo del semestre en curso. A continuación, se describen algunas definiciones importantes:

3.1.1.1 ***Niveles académicos.*** Las asignaturas del programa de Ingeniería Industrial se clasifican en niveles académicos, estos niveles van desde el nivel 1 al 10. Para el ciclo profesional del programa los niveles académicos van desde el Nivel 3 hasta el 10.

3.1.1.2 ***Currículos.*** Un currículo es un conjunto de cursos del mismo nivel académico que se espera sean atendidos por el mismo grupo de estudiantes, esta definición es usada para que los estudiantes tengan más posibilidades de matricular sus materias sin que se presente interferencia horaria entre ellas.

3.1.1.3 ***Asignaturas y cursos.*** Las asignaturas a programar son las consignadas en el Plan de Estudios para el ciclo profesional de la carrera de Ingeniería Industrial, esto incluye las asignaturas obligatorias, las electivas y las de servicios.

- Asignaturas exigibles: Son aquellas que la Universidad considera, dentro del respectivo plan de estudios, como de obligatoria matrícula y aprobación.
- Asignaturas electivas: Son aquellas que la Universidad ha establecido en el respectivo plan de estudios, para coadyuvar a la formación profesional.

- Asignaturas de servicio: Son aquellas asignaturas del respectivo plan de estudios que son requeridas por otros programas académicos y compartirán recursos con las asignaturas exigibles y electivas.

Además, de acuerdo con la modalidad del proceso enseñanza-aprendizaje, las asignaturas se clasifican en:

- Teóricas: Son aquellas en las cuales el proceso enseñanza-aprendizaje se realiza por medio de exposiciones del profesor, con participación de los estudiantes.
- Prácticas: Son aquellas en las cuales el proceso enseñanza-aprendizaje lo realiza el estudiante aplicando los conocimientos teóricos, bajo la dirección del profesor.
- Teórico-prácticas: Son aquellas en las cuales el proceso de enseñanza-aprendizaje se realiza combinando las dos modalidades anteriores de clase.

Cada asignatura tiene una cantidad de cursos a programar definidos por el Coordinador Académico del programa. Además, cada una de ellas, posee una intensidad académica medida en horas semanales, que para este caso oscila entre 3 y 5 horas semanales. Esta intensidad se traduce en la cantidad de eventos a programar, tal como se definió la sección 1.3.4.

3.1.1.4 ***Profesores.*** Son el personal de tiempo completo, medio tiempo o cátedra contratado por la institución para impartir una o más asignaturas. La programación de asignaturas para los profesores depende su clasificación ya sea tipo planta, cátedra o estudiante de posgrado.

- Profesor Planta: Desempeña la función de dirección de asignaturas bajo la modalidad de profesor de tiempo completo. Respecto a la programación de asignaturas, al profesor planta se le permite definir las asignaturas y cantidad de cursos por asignatura que puede impartir.
- Profesor Catedra: Desempeña la función de dirección de asignaturas de manera temporal. Su vinculación a la UIS se hace mediante un contrato laboral por el tiempo que se requiera. Para la programación de asignaturas, el profesor propone las asignaturas que puede impartir.
- Estudiante de posgrado: Lleva acabo la labor de contraprestación bajo la modalidad de Docencia directa en pregrado, previo acuerdo con el Consejo de Escuela, para impartir un curso de una asignatura con máximo una intensidad semanal de 6 horas.

Además, debe tenerse en cuenta que existe un número máximo de cursos que se pueden asignar a un profesor, el cual está definido por la Dirección de Escuela. Finalmente, los profesores de EEIE tienen una disponibilidad horaria semanal en la cual pueden impartir sus asignaturas.

3.1.1.5 ***Espacio físico.*** El edificio de la Escuela de Estudios Industriales y Empresariales está compuesto por aulas de clase, aulas talleres, laboratorio, sala de cómputo, sala de maestría, oficina de profesores y dependencias administrativas. Asimismo, cuenta con dos auditorios, el auditorio Guillermo Camacho Caro con capacidad para 180 personas y el auditorio Enrique Daccarett con capacidad estimada para 220 personas.

En lo que respecta al espacio físico destinado a la programación de asignaturas, EEIE cuenta con los siguientes salones disponibles: 28 aulas de clase, 1 laboratorio de Ingeniería de Calidad y una sala de cómputo.

3.1.1.6 ***Periodos y franjas horarias.*** Los periodos corresponden a los espacios de tiempo en los que se divide un día y generalmente, tienen una duración de una hora. Las franjas horarias corresponden a la totalidad de los periodos disponibles durante una semana.

En EEIE se dispone de 16 periodos diarios comprendidos entre las 6:00 am y 10:00 pm y 70 franjas horarias comprendidas entre el día lunes a las 6:00 am hasta el día viernes a las 12:00 m.

3.1.2 **Requerimientos y necesidades.** Los requerimientos y necesidades establecen el tipo de horarios que se busca producir. A continuación, se definen los requisitos que deben satisfacerse al momento de programar las asignaturas para el caso de EEIE.

- Intensidad horaria

  Todas las asignaturas y sus correspondientes cursos deben ser programadas según la intensidad horaria semanal establecida para estas.

- Interferencia horaria

  Los eventos (horas de clase semanales) de un curso deben ser programados de tal forma que no haya conflicto entre ellas, es decir, deben programarse en franjas diferentes.

  Los cursos asignados a un mismo profesor no pueden cruzarse entre sí, dado que este no podría impartir alguno de ellos.

Un salón solo puede albergar un curso durante una franja horaria, es decir, no se pueden programar dos cursos a la misma hora en un mismo salón.

Las asignaturas pertenecientes a un mismo nivel académico deben programarse en franjas diferentes con el fin de que no exista conflicto entre ellas.

- Asignación de profesores

Los profesores solo pueden ser asignados a asignaturas/cursos que se haya definido están en capacidad de impartir.

Cada curso debe tener asignado un único profesor que imparta cada uno de los eventos del curso.

A los profesores de tipo Planta y estudiantes de Posgrado se les debe asignar la cantidad de cursos por asignatura que se hayan acordado.

Ningún profesor puede impartir más de 4 cursos.

Ningún profesor puede impartir más de 3 cursos de una misma asignatura.

La disponibilidad horaria semanal definida por cada profesor debe ser respetada en la medida de lo posible.

- Asignación de franjas y periodos

Los cursos deben ser programados en sesiones diarias de 2 o 3 horas consecutivas.

Las sesiones de un curso asignado en dos días diferentes deben iniciar a la misma hora, siempre que sea posible.

Debe existir por lo menos un día de descanso entre las sesiones de cada curso.

- Asignación de salones

Los cursos deben ser asignados en salones que cumplan con los requerimientos exigidos por la asignatura a la que pertenecen.

Cada curso debe ser asignado en un único salón, a menos que los requisitos de la asignatura exijan lo contrario.

La programación de asignaturas debe hacerse buscando reducir la cantidad de salones usados.

- Paralelismo

Los cursos pertenecientes a la asignatura de Procesos Industriales deben asignarse paralelamente, es decir, en las mismas franjas, pero diferentes salones.

- Ingeniería de Calidad

La asignatura Ingeniería de Calidad es una asignatura compuesta por una sesión de aula y una sesión de laboratorio. Para esta asignatura se requiere que las sesiones de aula de cada par de cursos sean programadas en el mismo salón, en las mismas franjas y con el mismo profesor, contradiciendo los requerimientos de Interferencia horaria.

3.1.3 **Relajación de los requerimientos y necesidades.** Debido al alto grado de dificultad que conlleva modelar, y posteriormente resolver el problema de programación de asignaturas, se ha decidido relajar algunas características descritas en la sección anterior. En este apartado se definen las modificaciones que se realizan sobre el problema original:

- Intensidad horaria

Las asignaturas de servicio no son programadas.

Las asignaturas a programar son consideradas como asignaturas de modalidad teórica.

- Interferencia horaria

Dado que, en algunos casos es imposible programar todas las asignaturas de un nivel académico sin que se presenten interferencias entre estas, se ha optado por agrupar los cursos de un mismo nivel en currículos. Cada currículo, está formado por un curso de cada asignatura

del mismo nivel. De manera que, los cursos pertenecientes al mismo currículo deben programarse en franjas diferentes con el fin de que no exista conflicto entre ellas.

Las asignaturas que requieren el laboratorio de Ingeniería de Calidad o la sala de cómputo no deben presentar interferencias entre ellas.

- Asignación de salones

Solamente existen aulas de clase para realizar la programación.

El laboratorio de Ingeniería de Calidad y la sala de cómputo están excluidos de la programación de asignaturas. Sin embargo, las asignaturas que requieren estos recursos son programadas en su totalidad en aulas de clase.

- Ingeniería de Calidad

Debido a las dificultades que se describieron anteriormente respecto a esta asignatura. Para poder realizar su programación se ha optado por fusionar el par de cursos que comparten profesor, salón y franjas; de esta manera, se forma un curso de dos sesiones de laboratorio y una sesión de aula.

3.1.4 **Restricciones del problema.** Una vez considerandos los requerimientos y necesidades presentados en el apartado 3.1.2 y las modificaciones planteadas en el apartado 3.1.3 se pueden resumir las restricciones del problema en la siguiente tabla:

Tabla 4.
Restricciones del UCTP en EEIE

Restricción	Descripción
R1	Se deben programar los horarios de tal forma que se usen la menor cantidad de salones posibles.
R2	Las asignaturas a programar y sus correspondientes cursos deben ser asignados según la intensidad horaria semanal establecidas para estas.
R3	Los eventos (horas de semanales) de un curso deben ser programados de tal forma que no haya conflicto entre ellas, es decir, deben programarse en franjas diferentes.
R4	Los cursos asignados a un mismo profesor no pueden cruzarse entre sí, dado que este no podría impartir alguno de ellos.
R5	Un salón solo puede albergar un curso durante una franja horaria, es decir, no se pueden programar dos cursos a la misma hora en un mismo salón.
R6	Los cursos pertenecientes al mismo currículo deben programarse en franjas diferentes con el fin de que no exista conflicto entre ellas.
R7	Las asignaturas que requieren el Laboratorio de Ingeniería de Calidad o la Sala de Cómputo no deben presentar interferencias entre ellas.
R8	Los profesores solo pueden ser asignados a asignaturas/cursos que se haya definido están en capacidad de impartir.
R9	Cada curso debe tener asignado un único profesor que imparta cada uno de los eventos del curso.
R10	A los profesores de tipo Planta y estudiantes de Posgrado se les debe asignar la cantidad de cursos por asignatura que se hayan acordado.
R11	Ningún profesor puede impartir más de 4 cursos.
R12	Ningún profesor puede impartir más de 3 cursos de una misma asignatura
R13	La disponibilidad semanal definida por cada profesor debe ser respetada.
R14	Cada curso debe ser asignado en un único salón
R15	Las asignaturas deben ser programadas en sesiones diarias de 2 o 3 horas consecutivas.
R16	Las sesiones de un curso asignado en dos días diferentes deben iniciar a la misma hora.
R17	Debe existir por lo menos un día de descanso entre sesiones del mismo curso.
R18	Los cursos pertenecientes a la asignatura de Procesos Industriales deben asignarse paralelamente, es decir, en las mismas franjas, pero diferentes salones.

3.2 Formulación matemática del UCTP

Basados en lo descrito en la sección 3.1 es posible plantear la programación de asignaturas del programa de Ingeniería Industrial en EEIE como un problema de programación lineal entera mixta – PLEM. A continuación, se definen los elementos necesarios para la solución del problema.

3.2.1 Entradas

$i = \{1,2,3, \dots, c\}$ Cursos (c) que deben ser programados.

$j = \{1,2,3, \dots, t\}$ Profesores (t) disponibles.

$k = \{1,2,3 \dots, s\}$ Salones o aulas de clases (s) disponibles.

$l = \{1,2,3 \dots, d\}$ Días (d) de la semana.

$m = \{1,2,3 \dots, p\}$ Periodos (p) disponibles

$Q_h = \{1,2,3, \dots, h\}$ Currículos.

$G_e = \{1,2,3, \dots, e\}$ Asignaturas.

$i \cap Q_h$: Subconjunto creado a partir de los conjuntos Cursos y Currículos para identificar qué cursos pertenecen a cada currículo.

$i \cap G_e$: Subconjunto creado a partir de los conjuntos Cursos y Asignaturas para identificar qué cursos pertenecen a cada asignatura.

$iLAB$: Subconjunto que contiene los cursos de las asignaturas que deben programarse en el laboratorio de Ingeniería de calidad.

$iCOM$: Subconjunto que contiene los cursos de las asignaturas que deben programarse en la sala de cómputo.

N_i: Número de horas semanales que el curso i debe ser programado.

A_j: Número de cursos que deben impartir los profesores planta.

B_{ij}: Matriz de disponibilidad, que tiene valor 1 si el curso i puede ser impartida por el profesor j .

V_{jlm}: Matriz de disponibilidad, que tiene valor 1 si el profesor j está disponible para impartir el día l en el periodo m.

O_k: Vector de coeficientes, que tiene un valor entero [1,s] y penaliza asignar el salón k.

Una vez definidos los elementos que intervienen en la formulación, se pasa a definir las variables, la función objetivo y las restricciones que garantizaran la factibilidad del horario.

3.2.2 Variables

$$X_{ijklm} = \begin{cases} 1 \text{ si el profesor j enseña el curso i en el salón k el dia l en el periodo m} \\ 0 \text{ si no} \end{cases}$$

$$W_{ij} = \begin{cases} 1 \text{ si el profesor j enseña la curso i} \\ 0 \text{ si no} \end{cases}$$

$$H_{ik} = \begin{cases} 1 \text{ si el curso i se enseña en el salón k} \\ 0 \text{ si no} \end{cases}$$

$$Y_{il} = \begin{cases} 1 \text{ si el curso i se enseña el dia l} \\ 0 \text{ si no} \end{cases}$$

$E_{il} \in N$ Primer periodo asignado al curso i el dia l

$F_{il} \in N$ Ultimo periodo asignado al curso i el dia l

3.2.3 **Modelo del problema en PLEM.** El planteamiento se hace considerando una función mono-objetivo sujeta a 25 ecuaciones agrupadas en 18 restricciones.

R1. El objetivo del modelo es programar los horarios de tal forma que se usen la menor cantidad de salones posibles.

$$Minimizar\ Z = \sum_{i=1}^{C} \sum_{j=1}^{T} \sum_{k=1}^{S} \sum_{l=1}^{D} \sum_{m=1}^{P} O_k * X_{ijklm} \qquad (1)$$

Sujeto a:

R2. Las asignaturas a programar y sus correspondientes cursos deben ser asignados según la intensidad horaria semanal establecidas para estas.

$$\sum_{j=1}^{T}\sum_{k=1}^{S}\sum_{l=1}^{D}\sum_{m=1}^{P} X_{ijklm} = N_i \qquad \forall i \qquad (2)$$

R3. Los eventos (horas de semanales) de un curso deben ser programados de tal forma que no haya conflicto entre ellas, es decir, deben programarse en franjas diferentes.

$$\sum_{k=1}^{S}\sum_{j=1}^{T} X_{ijklm} \leq 1 \qquad \forall i\ \forall l\ \forall m \qquad (3)$$

R4. Los cursos asignados a un mismo profesor no pueden cruzarse entre sí, dado que este no podría impartir alguno de ellos.

$$\sum_{i=1}^{C}\sum_{k=1}^{S} X_{ijklm} \leq 1 \qquad \forall j\ \forall l\ \forall m \qquad (4)$$

R5. Un salón solo puede albergar un curso durante una franja horaria, es decir, no se pueden programar dos cursos a la misma hora en un mismo salón.

$$\sum_{i=1}^{C}\sum_{j=1}^{T} X_{ijklm} \leq 1 \qquad \forall k\ \forall l\ \forall m \qquad (5)$$

R6. Los cursos pertenecientes al mismo currículo deben programarse en franjas diferentes con el fin de que no exista conflicto entre ellas.

$$\sum_{k=1}^{S}\sum_{j=1}^{T}\sum_{i\cap Q_h} X_{ijklm} \leq 1 \qquad \forall Q_h\ \forall l\ \forall m \qquad (6)$$

R7. Las asignaturas que requieren el laboratorio de Ingeniería de Calidad o la sala de computo no deben presentar interferencias entre ellas.

$$\sum_{k=1}^{S}\sum_{j=1}^{T}\sum_{iLAB} X_{ijklm} \leq 1 \qquad \forall l\ \forall m \qquad (7)$$

$$\sum_{k=1}^{S}\sum_{j=1}^{T}\sum_{iCOM} X_{ijklm} \leq 1 \qquad \forall l\ \forall m \qquad (8)$$

R8. Los profesores solo pueden ser asignados a asignaturas/cursos que se haya definido están en capacidad de impartir.

$$W_{ij} \leq B_{ij} \qquad \forall i\ \forall j \qquad (9)$$

R9. Cada curso debe tener asignado un único profesor que imparta cada uno de los eventos del curso.

$$\sum_{k=1}^{S}\sum_{l=1}^{D}\sum_{m=1}^{P} X_{ijklm} - N_i * W_{ij} = 0 \qquad \forall i\ \forall j \qquad (10)$$

$$\sum_{j=1}^{T} W_{ij} = 1 \qquad \forall i \qquad (11)$$

R10. A los profesores de tipo Planta y estudiantes de Posgrado (PTC) se les debe asignar la cantidad de cursos por asignatura que se hayan acordado.

$$\sum_{i=1}^{C} W_{ij} = A_j \qquad j = \{1, \dots, PTC\} \qquad (12)$$

R11. Ningún profesor puede impartir más de 4 cursos.

$$\sum_{i=1}^{C} W_{ij} \leq 4 \qquad j = \{1,2,3, \dots, t\} \qquad (13)$$

R12. Ningún profesor puede impartir más de 3 cursos de una misma asignatura.

$$\sum_{i \cap G_e} W_{ij} \leq 3 \qquad j = \{12,13,14, \dots., t\} \quad \forall G_e \qquad (14)$$

R13. La disponibilidad semanal definida por cada profesor debe ser respetada.

$$\sum_{i=1}^{C}\sum_{k=1}^{S} X_{ijklm} \leq V_{jlm} \qquad \forall j\ \forall l\ \forall m \qquad (15)$$

R14. Cada curso debe ser asignado en un único salón

$$\sum_{l=1}^{D}\sum_{m=1}^{P}\sum_{j=1}^{T} X_{ijklm} - N_i * H_{ik} = 0 \qquad \forall i\ \forall k \qquad (16)$$

$$\sum_{k=1}^{S} H_{ik} = 1 \qquad \forall i \qquad (17)$$

R15. Las asignaturas deben ser programadas en sesiones diarias de 2 o 3 horas consecutivas.

$$\sum_{j=1}^{T}\sum_{k=1}^{S}\sum_{m=1}^{P} X_{ijklm} - 2Y_{il} \geq 0 \qquad \forall i\ \forall l \qquad (18)$$

$$\sum_{j=1}^{T}\sum_{k=1}^{S}\sum_{m=1}^{P} X_{ijklm} - 3Y_{il} \leq 0 \qquad \forall i\ \forall l \qquad (19)$$

$$E_{il} - (P+1) + (P+1-m)X_{ijklm} \leq 0 \qquad \forall i\ \forall j\ \forall k\ \ \forall l\ \forall m \quad (20)$$

$$F_{il} - m * X_{ijklm} \geq 0 \qquad \forall i\ \forall j\ \forall k\ \ \forall l\ \forall m \quad (21)$$

$$F_{il} - E_{il} + Y_{il} - \sum_{m=1}^{P} X_{ijklm} \leq 0 \qquad \forall i\ \forall j\ \forall k\ \ \forall l \qquad (22)$$

R16. Las sesiones de un curso asignado en dos días diferentes deben iniciar a la misma hora.

$$E_{il} - E_{i,l+1} = 0 \qquad \forall i\ \ l \in \{1,2,3,4\} \qquad (23)$$

R17. Debe existir por lo menos un día de descanso entre sesiones del mismo curso.

$$Y_{il} + Y_{i,l+1} \leq 1 \qquad \forall i\ \ l \in \{1,2,3,4\} \qquad (24)$$

R18. Los cursos pertenecientes a la asignatura de Procesos Industriales $C_{P(i)}$ deben asignarse paralelamente, es decir, en las mismas franjas, pero diferentes salones.

$$\sum_{k=1}^{S}\sum_{j=1}^{T} X_{C_{p1}jklm} - \sum_{k=1}^{S}\sum_{j=1}^{T} X_{C_{p2}jklm} = 0 \qquad \forall l\ \forall m \qquad (25)$$

4. Adaptación del algoritmo HGATS al problema específico de EEIE

La definición del problema de estudio de la Escuela de Estudios Industriales y Empresariales de la Universidad Industrial de Santander se encuentra en la sección 3.1, este difiere a la definición del problema planteado por los autores Yang y Jat (2011) el cual se detalla en el apartado 1.3.6 en los siguientes puntos:

- Las restricciones duras y suaves (sección 3.1.4) dependen de las necesidades y requerimientos de cada institución. En este caso, se tienen 14 restricciones duras y 4 suaves, a diferencia del problema de referencia tratado por los autores el cual contiene 5 restricciones duras y 3 suaves.
- Consecuentemente, con el aumento de las restricciones se hace necesario el uso de otras estructuras para abordar el problema.
- Los eventos están agrupados en sesiones de 2 y 3 periodos consecutivos, mientras que los eventos del algoritmo original tienen una longitud de 1 periodo. Por este motivo, el proceso de iteración en los diferentes operadores se hace por medio de bloques.

- Debido a que solo se programan aulas de clase, no es necesario manejar la lista de salones idóneos y las matrices asociadas a estos.
- Los estudiantes y las estructuras de datos relacionadas no son considerados dentro de nuestro problema.
- En lugar de la matriz evento-disponibilidad se ha decidido introducir la matriz profesor-disponibilidad.
- Se ha incluido una Heurística de Asignación de Profesores que opera sobre la instancia inicial, antes de comenzar el proceso de iteración.
- Se ha incluido una Heurística de Estabilidad de Salones que opera sobre la mejor solución obtenida por el Algoritmo HGATS con el objetivo de satisfacer la restricción R14.
- En el algoritmo modificado, los vecindarios usados son los generados por las estructuras N1, N2, N3 y N5.
- El Algoritmo de Coincidencia usado en los Operadores de Búsquela Local 1 y 2 no es requerido en la programación de salones.
- La estructura de vecindario N5 que, en el algoritmo original es usado para tratar de cumplir las restricciones de secuencia D5, es modificado para atender las restricciones de paralelismo R18.
- La Estructura de Memoria (MEN) y, por lo tanto, el Algoritmo de Búsqueda Guiada por MEM no son usadas; esto se debe a que todos los eventos del problema en EEIE tienen franjas restringidas por la disponibilidad del profesor asignado.
- En la fase de Búsqueda Tabú, no se permite remover los eventos que generan violaciones en las restricciones duras.

4.1 Estructuras de datos

A continuación, se describen como están organizados los datos requeridos por el algoritmo. Los valores numéricos sirven como referencia y no representan los datos de las instancias a evaluar.

4.1.1 **Datos generales.** Se tiene una tabla con los datos generales del problema, esta tabla contiene información del total de asignaturas a programar, el número de profesores, el número de currículos en los que están agrupados los cursos, el total de cursos, la cantidad de aulas de clase disponibles para la programación, así como también, los días, periodos, y franjas disponibles.

Tabla 5.
Datos generales

Asignaturas	Docentes	Currículos	Cursos	Salones	Días	Periodos	Franjas
36	57	21	120	12	5	16	70

4.1.2 **Asignaturas.** La siguiente tabla presenta información relacionada a las asignaturas, nivel académico al que pertenecen, número de cursos que deben programarse por asignatura, intensidad horaria de cada una de ellas, y cantidad de profesores que pueden impartir la asignatura.

Tabla 6.
Estructura de datos de las asignaturas

Asignatura (G)	Nivel	Cursos	I. Horaria	Profesores
1	3	4	4	2
2	4	4	4	2
5	6	3	3	1
6	6	5	4	3
7	6	9	3	5
⋮	⋮	⋮	⋮	⋮
35	10	1	3	1
36	10	1	4	1

4.1.3 **Currículos.** La estructura de los currículos se define como una lista de vectores, donde cada vector contiene los cursos que no deben presentar interferencia horaria entre sus eventos.

4.1.4 **Asignatura-Profesor.** La matriz asignatura-profesor permite identificar las asignaturas que puede impartir cada profesor. Cada elemento de la matriz toma el valor de 1 si el profesor está capacitado para impartir la asignatura y 0 sino.

Tabla 7.
Estructura asignatura-profesor

	Profesor					
Asignatura	T1	T2	T3	…	T56	T57
G1	1	1	0	…	1	0
G2	1	0	0	…	0	0
G3	0	0	0	…	1	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮
G35	0	1	0	…	0	0
G36	1	0	1	…	0	0

4.1.5 **Profesores prioritarios.** Los profesores clasificados como profesores Planta y Estudiantes de Posgrado se asignan de forma diferente a los profesores Catedra. Por ello, para estos profesores se requiere una matriz adicional.

Tabla 8.
Estructura Prioridad profesor

	Asignatura					
Profesor	G1	G2	G3	…	G35	G36
T1	1	2	0	…	0	1
T2	1	0	0	…	2	0
T3	0	0	0	…	0	3
⋮	⋮	⋮	⋮	⋮	⋮	⋮
T12	0	1	0	…	0	1
T13	1	0	2	…	0	0

Esta matriz se usa para asignar la cantidad de cursos por asignatura requerida para profesores Planta y Estudiantes de Posgrado. Cada elemento dentro de la matriz representa el número de cursos por asignatura que debe ser asignado a cada profesor.

4.1.6 **Disponibilidad-Profesor.** La siguiente matriz representa las franjas en las que está disponible cada uno de los profesores a lo largo de la semana.

Tabla 9.
Disponibilidad de los profesores

	Franjas horarias														
Profesor	1	2	3	4	5	6	7	…	64	65	66	67	68	69	70
T1	1	1	0	0	1	1	1	…	1	0	0	0	1	1	1
T2	1	1	1	0	0	1	1	…	1	1	0	0	0	1	1
T3	1	1	1	1	0	0	1	…	1	1	1	0	0	0	1
T4	0	0	1	1	1	0	0	…	0	1	1	1	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
T55	1	0	1	0	1	1	1	…	0	1	1	1	0	0	1
T56	0	0	0	1	1	0	1	…	1	0	1	1	1	0	0
T57	1	1	0	1	0	1	1	…	0	0	0	0	1	1	1

Los valores de la matriz toman valor de 1 si el profesor está disponible para impartir sus asignaturas en dicha franja y 0 sino.

4.1.7 **Salón-Asignatura.** Dado que es necesario saber qué asignaturas usan la sala de cómputo y el Laboratorio de Ingeniería de Calidad se requiere el uso de la siguiente estructura.

Tabla 10.
Estructura salones-asignaturas

	Asignatura					
Tipo de salón	G1	G2	G3	…	G35	G36
Aula de clase	4	2	2	…	4	4
Sala de computo	0	2	0	…	0	0
Laboratorio de calidad	0	0	4	…	0	0

En esta tabla, cada valor numérico representa el número de horas por asignatura que debe asignarse en los diferentes tipos de salones.

4.2 Representación de la solución

La representación usada varía un poco respecto a la propuesta por el autor. En este caso, la solución está representada por una lista de pares franja-salón compuesta por todas las combinaciones de salón y franja posibles, en donde el índice los primeros n pares es el identificador de cada evento y los pares numerados después del evento n, son pares que no han sido asignados.

Respecto a los eventos, cada evento guarda información de la asignatura, el curso al que pertenece, y el profesor que atiende el evento.

Figura 11. Representación del gen

Siendo, G el conjunto de asignaturas, C_G el número de cursos por asignatura y N_G la intensidad horaria semanal de cada asignatura. El total de eventos n a programar está definido por:

$$Total\ eventos = \sum_{i=1}^{G} C_G(i) * N_G(i)$$

Tabla 11.
Representación de la solución

	$\boldsymbol{franja}$	$\boldsymbol{salón}$
$e1$	1	1
$e2$	1	2
$e3$	2	3
⋮	⋮	⋮
$e(n-2)$	1	4
$e(n-1)$	5	6
$e(n)$	2	4
$e(n+1)$	4	6
$e(n+2)$	6	4
⋮	⋮	⋮
$e(franjas * salones)$	1	5

En la solución presentada, el evento 1 ha sido asignado en la franja 1 del salón 1, el evento 2 en la franja 1 del salón 2; mientras que, el salón 6 en la franja 4 está desocupado. De esta forma, se garantiza el cumplimiento de las restricciones R2 y R5.

4.3 Función de aptitud

Se establece una función de aptitud con el fin de minimizar las violaciones de las restricciones duras y suaves. La función de aptitud $F(s)$ para la solución s, se define una función como la suma del número de violaciones de restricciones duras $\#hcv$ y restricciones suaves $\#scv$.

$$F(s) = \#hcv * C + \#svc$$

Restricciones duras: Las restricciones clasificadas como duras que deben ser satisfechas en su totalidad para considerar que la solución es factible, son las restricciones R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R15 y R18. De este grupo, las restricciones R2 y R5 son garantizadas en su totalidad debido a la representación de la solución, pues siempre garantiza que todos los eventos sean programados y que no exista interferencia horaria para los salones. Adicionalmente, las restricciones de la R8 a la R12 son garantizadas en su totalidad a través de la Heurística de Asignación de Profesores la cual, vincula aleatoriamente un curso con un profesor, mientras satisface dichas restricciones.

Restricciones suaves: Estas restricciones miden la calidad de la solución, una solución es de mejor calidad si, en medida de lo necesario, satisface estas restricciones. En el caso de EEIE, las restricciones suaves son las R1, R14, R16 y R17.

El valor de constante C es de 1000. De esta forma se garantiza qué durante el proceso de búsqueda, se priorice las restricciones duras sobre las suaves.

Finalmente, la función de aptitud para el caso de EEIE queda expresado como:

$$F(s) = 1000 * \#hcv + \#svc$$

$$\#hcv = f(R3) + f(R4) + f(R6) + f(R7) + f(R13) + f(R15) + f(R18)$$

$$\#svc = f(R1) + f(R14) + f(R16) + f(R17)$$

La valoración de cada restricción se lleva a cabo de la siguiente manera para las restricciones duras:

- $f(R3)$ penaliza con el valor de 1 cada vez que dos eventos del mismo curso presentan interferencia horaria entre ellos.
- $f(R4)$ penaliza con el valor de 1 cada vez que dos eventos asignados al mismo profesor presentan interferencia horaria entre ellos.
- $f(R6)$ penaliza con el valor de 1 cada vez que dos eventos que pertenecen al mismo currículo presentan interferencia horaria entre ellos.
- $f(R7)$ penaliza con el valor de 1 cada vez que dos eventos que usan el Laboratorio de Ingeniería de Calidad o la Sala de Cómputo presentan interferencia entre sí.
- $f(R13)$ penaliza con el valor de 1 cada vez que se programa un evento fuera de la disponibilidad semanal del profesor asignado.
- $f(R15)$ define la penalidad como el número periodos vacíos entre eventos de un curso programado en un día. Adicionalmente, penaliza con el valor de 1 si un curso con más de tres eventos es programado en su totalidad en un único día.
- $f(R18)$ penaliza la diferencia entre los periodos asignados a las sesiones paralelas de los cursos de Procesos Industriales.

Mientras que, para las restricciones blandas, se calcula la penalización de la siguiente manera:

- $f(R1)$ penaliza con el uso ineficiente de los salones que se dispone. Por ejemplo, si se deben programar 400 eventos en un semestre, idealmente, deberían usarse únicamente los primeros 6 salones para asignar dichos eventos (considerando 70 franjas semanales por salón). De esta manera, si hay asignaciones en franjas de otros salones se considera una penalización calculada como la razón entre las franjas asignadas en dichos salones y el número franjas semanales.
- $f(R14)$ penaliza con el valor de 1 si los eventos de un curso son asignados en más de un salón. Por ejemplo, si un curso con 5 eventos es asignado en 5 salones diferentes, se calcula una penalidad de 4 unidades en dicho caso.
- $f(R16)$ penaliza la diferencia entre los primeros periodos asignados a sesión perteneciente al mismo curso. Por ejemplo, si un curso tiene sesiones en el periodo 3 y el periodo 6, en días diferentes, la penalización correspondiente en dicho caso sería de 3 unidades.
- $f(R17)$ penaliza con el valor de 1 cada vez que las sesiones de un curso se programen en días consecutivos.

La solución es óptima cuando la suma de todas las penalidades consideradas en la función de aptitud da como resultado el valor de cero.

4.4 Adaptación del algoritmo

Se inicia definiendo la instancia y los valores de los parámetros involucrados. Para la primera etapa del algoritmo HGATS, los parámetros de tamaño de población, cantidad de generaciones,

probabilidad de mutación y número de pasos guían la evolución del algoritmo GSGA; mientras que, en la segunda fase, el algoritmo TS está regido por los parámetros de porcentaje de vecindario, tamaño de la lista Tabú y cantidad de iteraciones.

En GSGA, inicialmente, se ejecuta la Heurística de Asignación de Profesores, la cual vincula profesores y cursos; luego, el Algoritmo Genético construye la población de soluciones iniciales usando el algoritmo Inicialización de Individuos y los operadores de Búsqueda Local 1 y 2; seguidamente, se pasa al proceso de evolución, allí, los operadores de Cruzamiento y Mutación generan una nueva solución, que es mejorada por los operadores de Búsqueda Local 1 y 2; después, se selecciona la peor solución de la población y se remplaza con la nueva solución generada. Este procedimiento se repite hasta alcanzar el número de generaciones definido. Finalmente, se aplica la Heurística de Estabilidad de Salones sobre la mejor solución obtenida y se evalúa su aptitud. Si la solución es óptima, el algoritmo se detiene.

El algoritmo TS se ejecuta solamente si la mejor solución generada en la primera fase no es óptima, es decir si $F(s) > 0$. En esta fase, una heurística de Búsqueda Tabú es aplicada, dicha heurística hace uso N1 y N2 como estructuras de vecindario para mover la solución en el espacio de búsqueda. Durante cada iteración, se calcula la penalización de cada uno de los posibles movimientos y se hace el mejor de ellos. Con cada movimiento la lista tabú es actualizada y se restringen algunos movimientos para futuras iteraciones. Una vez se cumple la cantidad de iteraciones definida, la heurística de Búsqueda Tabú arroja la mejor solución encontrada hasta el momento y aplica la Heurística de Estabilidad de Salones sobre la solución.

A continuación, se presenta el seudocódigo del algoritmo HGATS modificado para el caso EEIE.

Se inicia definiendo la instancia y los valores de los parámetros involucrados. Para la primera etapa del algoritmo HGATS, los parámetros de tamaño de población, cantidad de generaciones, probabilidad de mutación y número de pasos guían la evolución del algoritmo GSGA; mientras que, en la segunda fase, el algoritmo TS está regido por los parámetros de porcentaje de vecindario, tamaño de la lista Tabú y cantidad de iteraciones.

En GSGA, inicialmente, se ejecuta la Heurística de Asignación de Profesores, la cual vincula profesores y cursos; luego, el Algoritmo Genético construye la población de soluciones iniciales usando el algoritmo Inicialización de Individuos y los operadores de Búsqueda Local 1 y 2; seguidamente, se pasa al proceso de evolución, allí, los operadores de Cruzamiento y Mutación generan una nueva solución, que es mejorada por los operadores de Búsqueda Local 1 y 2; después, se selecciona la peor solución de la población y se remplaza con la nueva solución generada. Este procedimiento se repite hasta alcanzar el número de generaciones definido. Finalmente, se aplica la Heurística de Estabilidad de Salones sobre la mejor solución obtenida y se evalúa su aptitud. Si la solución es óptima, el algoritmo se detiene.

El algoritmo TS se ejecuta solamente si la mejor solución generada en la primera fase no es óptima, es decir si $F(s) > 0$. En esta fase, una heurística de Búsqueda Tabú es aplicada, dicha heurística hace uso N1 y N2 como estructuras de vecindario para mover la solución en el espacio de búsqueda. Durante cada iteración, se calcula la penalización de cada uno de los posibles

movimientos y se hace el mejor de ellos. Con cada movimiento la lista tabú es actualizada y se restringen algunos movimientos para futuras iteraciones. Una vez se cumple la cantidad de iteraciones definida, la heurística de Búsqueda Tabú arroja la mejor solución encontrada hasta el momento y aplica la Heurística de Estabilidad de Salones sobre la solución.

A continuación, se presenta el seudocódigo del algoritmo HGATS modificado para el caso EEIE.

```
1:   Entrada: Instancia del problema
2:   Definición de parámetros
     //Inicia algoritmo HGATS
3:   Ejecutar la Heurística de Asignación de Profesores // Vincula un curso con un
     Profesor

     //Inicia GSGA
4:   for i=1: tamaño de la población // Genera la población de soluciones
5:       Generar un individuo mediante el operador Inicializar Individuo
6:       Aplicar el operador de Búsqueda Local 1 sobre el individuo
7:       Aplicar el operador de búsqueda Local 2 sobre el individuo
8:       Evaluar la aptitud del individuo
9:   end
10:  while el número de generaciones no sea alcanzado // Evoluciona la población
11:      Generar un descendiente usando el operador de Cruzamiento
12:      Aplicar el operador de Mutación sobre el descendiente con cierta probabilidad
13:      Aplicar el operador de Búsqueda Local 1 sobre el descendiente
14:      Aplicar el operador de búsqueda Local 2 sobre el descendiente
15:      Reemplazar el peor individuo de la población por el descendiente
17:  end
18:  Seleccionar el mejor individuo 'S' de la población
19:  Aplicar Heurística de Estabilidad de Salones sobre 'S'
     // Finaliza GSGA
20:  if 'S' no es una solución optima
         // Inicia TS
21:      Aplicar la heurística de Búsqueda Tabú sobre 'S'
22:      Generar una solución óptima o mejorada 'SS'
23:      Aplicar Heurística de Estabilidad de Salones sobre 'SS'
24:      Salida: 'SS' es la mejor solución encontrada
         // Finaliza TS
25:  else
26:      Salida: 'S' es la solución óptima del problema
27:  end
28:  // Finaliza algoritmo HGATS
```

Figura 12. Algoritmo HGATS modificado

4.4.1 **Heurística de Asignación de Profesores.** Esta heurística se encarga de asignar aleatoriamente los profesores a los cursos mientras garantiza el cumplimiento de las restricciones:

R8. Los profesores solo pueden ser asignados a asignaturas/cursos que se haya definido están en capacidad de impartir.

R9. Cada curso debe tener asignado un único profesor que imparta cada uno de los eventos del curso.

R10. A los profesores de tipo Planta y estudiantes de Posgrado se les debe asignar la cantidad de cursos por asignatura que se hayan acordado.

R11. Ningún profesor puede impartir más de 4 cursos.

R12. Ningún profesor puede impartir más de 3 cursos de una misma asignatura.

El algoritmo inicia ordenando las asignaturas según la cantidad de profesores que hay disponibles para cada una; luego, selecciona la lista de posibles profesores que pueden impartir la asignatura. Si hay algún profesor tipo Planta o Estudiante de Posgrado que pueda impartir la asignatura, se procede a asignar la cantidad de cursos requeridos por el profesor para esa asignatura. Si hay más cursos por asignar se pasa a vincular a los profesores Cátedra garantizando que no puedan impartir más de 3 cursos de una misma asignatura, ni permitiendo que impartan más de 4 cursos en total.

```
1:   Entrada: Instancia del problema
2:   Ordenar las asignaturas según la cantidad de profesores que tienen disponibles
3:   for i=1: total de asignaturas
4:       Crear una lista TEMP de profesores que pueden impartir la asignatura i
5:       while existan cursos por asignar de la asignatura i
6:             if existen profesores Planta o Estudiantes de Posgrado en TEMP
7:                Asignar la cantidad de cursos requeridos por dichos profesores
8:             end
9:             if aún hay cursos sin asignar para la asignatura i
10:               Crear una lista TEMP2 a partir de TEMP con los profesores
                  catedra que no violan las restricciones R11 y R12
11:               Seleccionar aleatoriamente un profesor de TEMP2 y asignarlo
                  a uno de los cursos de la asignatura i
12:            end
13:      end
14:  end
15:  Salida: Tabla con pares ordenados de curso-profesor
```

Figura 13. Algoritmo de asignación de profesores

4.4.2 **Bloques.** Con el objetivo de limitar el espacio de búsqueda en regiones factibles, se ha optado por introducir el concepto de bloques. Para ello, se ha determinado que los eventos deben estar agrupados en bloques de 2 o 3 eventos consecutivos, dependiendo del número de eventos de cada curso. Si a un curso se le deben asignar 3 eventos, se construye un 1 bloque de tres eventos consecutivos, si tiene 4 eventos, se agrupan en 2 bloques de dos eventos consecutivos y si se tienen 5 eventos se crea un bloque de 2 eventos y uno de 3.

Adicionalmente, cuando se hace una asignación o movimiento de pares salón-franja estos deben estar agrupados en bloques de franjas consecutivas. Esto con el fin de facilitar el cumplimiento de la restricción dura R15.

4.4.3 **Inicializar Individuos.** Cada individuo representa una solución del problema. Para crear un individuo, se inicia generando dos matrices, la primera matriz, llamada cromosoma la cual, contiene la información relacionada a los eventos y la segunda matriz, llamada pares salón-franja, contiene las combinaciones de salones y franjas tal como se describió en la sección 4.2.

Luego se pasa a asignar cada par salón-franja a cada evento. Para ello, se inicia con los profesores, para cada profesor se genera un vector de cursos asignados y un vector de disponibilidad semanal. Ahora, para cada curso en dicho vector se identifican los eventos que no han sido asignados, mientras los eventos de un curso no sean asignados en su totalidad el algoritmo trata de asignarle un bloque aleatorio de pares salón-franja de dos o tres eventos consecutivos, de ser posible, dentro de la disponibilidad del profesor.

```
Entrada: Instancia del problema
Crear la matriz CROMOSOMA con los todos los eventos a ser programados
Crear la matriz PARES SALON-FRANJA con todas las combinaciones de salones y franjas posibles
for i=1: número de profesores
    Crear un vector CURSOS_DOCENTE con los cursos asignados al profesor i
    Guardar la disponibilidad del profesor i en un vector FD
    for j=1: tamaño del vector CURSOS_DOCENTE
        Contar cuantos eventos hay sin asignar el para el curso j
        while existan eventos sin asignar para el curso j
              Definir el tamaño del bloque que debe asignarse a los eventos del curso j
              Construir todos los bloques de franjas consecutivas posibles dentro del vector
              FD según el tamaño de bloque a asignar
              if hay más de un bloque de franjas consecutivas
                 Seleccionar un bloque aleatoriamente
                 if las franjas del bloque seleccionado están disponibles dentro de la matriz
                     PARES SALÓN-FRANJA
                     Asignar los primeros pares salón-franja que coincidan con las franjas del
                     bloque seleccionado
                 else
                      Eliminar las franjas no disponibles en la matriz PARES SALON-FRANJA del
                      vector FD
                 end
              else
                   Construir todos los bloques de franjas consecutivas posibles usando la matriz
                   PARES SALON-FRANJA según el tamaño de bloque a asignar
                   Seleccionar un bloque aleatoriamente
                   Asignar los primeros pares salón-franja que coincidan con las franjas del
                   bloque seleccionado
              end
        end
    end
end
Salida: Individuo
```

Figura 14. Algoritmo de Inicialización de Individuos

4.4.4 **Estructuras de vecindarios.** Para llevar a cabo los movimientos dentro de cada individuo, se usan 4 estructuras de vecindarios:

N1: Es un operador que intercambia los pares salón-franja asignados a un bloque de eventos por pares salón-franja no asignados.

N2: Es un operador que intercambia los pares salón-franja de dos bloques de eventos, del mismo tamaño, entre ellos.

N3: Es un operador que intercambia los pares salón-franja de tres bloques de eventos, del mismo tamaño, entre sí.

N5: Es un operador que toma bloques de pares salón-franja no asignados que comparten las mismas franjas y los intercambia con los pares salón-franja de los bloques de eventos que deben programarse paralelamente.

4.4.5 **Operador de Búsqueda Local 1.** El operador de búsqueda Local 1 (LS1) trabaja sobre todos los bloques de eventos de la solución. Este algoritmo, trabaja en dos fases, en la primera, se vale de las estructuras N1, N2 y N3 para intercambiar bloques eventos con el fin de obtener una solución factible. Si una solución factible es encontrada en esta fase, la segunda fase es ejecutada. La segunda fase opera de forma similar a la primera, sin embargo, en esta se consideran las violaciones de restricciones suaves. De esta forma se busca obtener una solución factible de buena calidad.

El algoritmo inicia verificando la factibilidad de cada bloque de eventos (un bloque de eventos es infactible si alguno de ellos viola alguna restricción dura). Si un bloque es infactible, se aplican

las estructuras N1, N2 y N3 ordenadamente hasta encontrar una mejora o hasta alcanzar cierto número de pasos. Para verificar si hay una mejora se usa una Evaluación Delta, tal como lo describen los autores del algoritmo en su trabajo. Una vez se evalúan todos los bloques, se calcula la factibilidad de la solución, si la solución obtenida sigue siendo infactible, el algoritmo finaliza, sino, se pasa a verificar la calidad de cada bloque de eventos. Si un bloque tiene una penalización en su calidad, se aplican las estructuras N1, N2 y N3 tal como se hizo anteriormente. Una vez evaluados todos los bloques, LS1 termina arrojando un Individuo potencialmente mejorado.

```
Entrada: Instancia del problema, Individuo/Descendiente, Parámetros
Crear bloques de eventos
for i=1: número de bloques creados
    verificar la factibilidad del bloque de eventos i
    if el bloque i es infactible
        Definir los posibles movimientos usando N1
        while no se cumpla el número de pasos definidos o no se mejore la factibilidad
              del bloque
              Generar una solución vecina
              Calcular Delta // evalúa la factibilidad del bloque antes y después del movimiento
              if Delta registra una mejora
                 La solución vecina sustituye al individuo/descendiente
              end
        end
        if N1 falla en mejorar la solución
           Definir los posibles movimientos usando N2
           while no se cumpla el número de pasos definidos o no se mejore la factibilidad
                 del bloque
                 Generar una solución vecina
                 Calcular Delta // evalúa la factibilidad del bloque antes y después del movimiento
                 if Delta registra una mejora
                    La solución vecina sustituye al individuo/descendiente
                 end
           end
           if N1 y N2 fallan en mejorar la solución
              Definir los posibles movimientos usando N3
              while no se cumpla el número de pasos definidos o no se mejore la factibilidad
                    del bloque
                    Generar una solución vecina
                    Calcular Delta // evalúa la factibilidad del bloque antes y después del movimiento
                    if Delta registra una mejora
                       La solución vecina sustituye al individuo/descendiente
                    end
              end
           end
        end
    end
end
if la solución es factible
   Ejecutar el operador de Búsqueda Local 1 (Parte 2)
end
Salida: Individuo/Descendiente mejorado
```

Figura 15. Operador de Búsqueda Local 1 (Parte 1)

```
1:   Entrada: operador de Búsqueda Local 1 (Parte 1)
2:   for i=1: número de bloques creados
3:       verificar la calidad del bloque de eventos i
4:       if el bloque i es viola alguna restricción suave
5:           Definir los posibles movimientos usando N1
6:           while no se cumpla el número de pasos definidos o no se mejore la aptitud
                 del bloque
7:                Generar una solución vecina
8:                Calcular Delta // evalúa la aptitud del bloque antes y después del movimiento
9:                if Delta registra una mejora
10:                  La solución vecina sustituye al individuo/descendiente
11:               end
12:          end
13:          if N1 falla en mejorar la solución
14:             Definir los posibles movimientos usando N2
15:             while no se cumpla el número de pasos definidos o no se mejore la aptitud
                      del bloque
16:                   Generar una solución vecina
17:                   Calcular Delta // evalúa la aptitud del bloque antes y después del movimiento
18:                   if Delta registra una mejora
19:                      La solución vecina sustituye al individuo/descendiente
20:                   end
21:             end
22:             if N1 y N2 fallan en mejorar la solución
23:                Definir los posibles movimientos usando N3
24:                while no se cumpla el número de pasos definidos o no se mejore la aptitud
                         del bloque
25:                      Generar una solución vecina
26:                      Calcular Delta // evalúa la aptitud del bloque antes y después del movimiento
27:                      if Delta registra una mejora
28:                         La solución vecina sustituye al individuo/descendiente
29:                      end
30:                end
31:             end
32:          end
33:      end
34:  end
35:  Salida: Individuo/Descendiente mejorado
```

Figura 16. Operador de Búsqueda Local 1 (Parte 2)

4.4.6 **Operador de Búsqueda Local 2.** El operador de Búsqueda Local 2 (LS2), intenta satisfacer la restricción de paralelismo R18, para ello, hace uso de la estructura de vecindario N5 y hace los movimientos siempre y cuando se presente una mejoría en la aptitud de la solución.

```
1:   Entrada: Instancia del problema, Individuo/descendiente generado por LS1, Parámetros
2:   Identificar los eventos que deben programarse paralelamente
3:   Definir los bloques de eventos paralelos
4:   Definir los posibles movimientos usando N5
5:   while no se alcance el número de pasos definidos o no se encuentre una mejora de
           aptitud en los bloques involucrados
6:         Generar una solución vecina
7:         Calcular Delta // evalúa la aptitud de los bloques antes y después del movimiento
8:         if Delta registra una mejora
9:            La solución vecina sustituye al individuo/descendiente
10         end
11:  end
12:  Salida: Individuo/descendiente posiblemente mejorado
```

Figura 17. Operador de Búsqueda Local 2

4.4.7 **Operador de Cruzamiento.** Este operador inicia seleccionando dos individuos de la población de soluciones como padres P1 y P2 a través del método Torneo de selección. En el proceso de cruce, el padre P1 es inicialmente, seleccionado como el descendiente. Luego, para cada bloque, se crea una solución vecina intercambiando los pares salón-franja del descendiente actual por los del padre P2 (mientras se garantiza que las restricciones R5 y R15 no sean violadas). Finalmente, el descendiente se actualiza con los pares salón-franja del padre P2, si existe una mejoría en la aptitud de la solución.

```
1:   Entrada: Población de soluciones
2:   Seleccionar a los padres P1 y P2 por la heurística de Torneo de Selección
3:   Ordenar los bloques de eventos
4:   Definir al padre P1 como descendiente
5:   for i=1: total de bloques
6:       Generar una nueva solución introduciendo los pares salón-franja del bloque i
         del padre P2 en el descendiente actual
7:       Reparar la nueva solución // evitar la creación de pares salón-franja duplicados
8:       if la nueva solución presenta una mejoría respecto al descendiente actual
9:          la nueva solución pasa a ser el descendiente
10:      end
11:  end
12:  Salida: Descendiente
```

Figura 18. Operador de cruzamiento

4.4.8 **Operador de Mutación.** El operador de mutación se encarga de mover aleatoriamente cada bloque de eventos con una probabilidad Pm usando las estructuras de vecindario N1, N2 y N3 para hacer los movimientos dentro de la solución.

```
1:   Entrada: Descendiente, Parámetros
2:   Ordenar los bloques de eventos
3:   Seleccionar aleatoriamente la estructura de vecindario a utilizar
     entre N1, N2, y N3
4:   if N1 o N2 o N3 es seleccionada
5:       for i=1: número de bloques
6:           Generar un aleatorio
7:           if el aleatorio es menor que la probabilidad de mutación Pm
8:             Definir los posibles movimientos usando N1 o N2 o N3
9:             Escoger un movimiento aleatoriamente
10:            Hacer el movimiento seleccionado
11:          end
12:      end
13:  end
14:  Salida: Descendiente mutado
```

Figura 19. Operador de mutación

4.4.9 **Heurística de Estabilidad de Salones.** Si la solución generada por la fase GSGA del algoritmo no logra satisfacer unos mínimos relacionados a la estabilidad de salones. Esta heurística es ejecutada para tratar de satisfacer la restricción suave R14 lo mejor que pueda. En este caso, los movimientos se hacen entre salones asignados a la misma franja, es decir, solamente se intercambian los salones. El procedimiento inicia en orden a los profesores. Para cada profesor, se identifican los cursos que tiene asignado; luego para cada curso se identifican las franjas asignadas. Si hay más de un salón asignado a dichas franjas se intenta ubicar las franjas en un único salón y se bloquean los pares salón-franja de dicho curso para que no sean intercambiados en futuros movimientos. Si no es posible asignar todas las franjas en un único salón, se pasa a asignar las franjas de los bloques de eventos del curso y una vez vinculados sus salones, se bloquean sus pares salón-franja.

```
Entrada: Mejor solución generada por GSGA/TS
for i=1: número de profesores
    Identificar los cursos asignados al profesor i
    for j=1: número de cursos
        Identificar los pares salón-franja asignados al curso j
        Intercambiar los pares salón-franja buscando asignar un único salón
        a los eventos del curso
        if el curso i logra ser ubicado en un único salón
            Bloquear pares salón-franja asignados a los eventos del curso
        else
            for k=1: número de bloques del curso j
                Identificar los pares salón-franja asignados al bloque k
                Intercambiar los pares salón-franja buscando asignar un único
                salón a los eventos del bloque k
                if el bloque k logra ser ubicado en un único salón
                    Bloquear pares salón-franja asignados al bloque k
                end
            end
        end
    end
end
Salida: Solución con salones estables
```

Figura 20. Heurística de Estabilidad de Salones

4.4.10 **Heurística de Búsqueda Tabú.** Una vez finalizada la fase GSGA del algoritmo, si la solución obtenida no es la óptima, una heurística de Búsqueda tabú opera sobre la mejor solución obtenida hasta el momento.

El algoritmo TS hace uso de las Estructuras vecinales N1 y N2 para generar los posibles movimientos de la solución. También, considera tabú un movimiento que involucre algún evento que se haya movido hace cierto número de iteraciones (tamaño de lista tabú). Además, aplica una vecindad variable que escoge aleatoriamente cierto porcentaje de la vecindad V, definida por N1 y N2. El algoritmo inicia creando una lista de eventos tabú vacía; mientras la cantidad de iteraciones definida no sea alcanzada, el algoritmo se encarga de crear todas las vecindades posibles permitidas por las estructuras N1 y N2 en cada iteración. Luego, realiza los V movimientos y calcula la aptitud de cada uno. Si algún movimiento realizado, genera una solución con mejor calidad que la solución actual, se guarda dicha solución como la solución actual y se

registran los eventos involucrados en el movimiento como tabús, en el caso contrario, se escoge la mejor solución no tabú como la solución actual y se registran los eventos involucrados en el movimiento como tabús. El algoritmo termina cuando se completa el número de iteraciones definido y arroja la mejor solución obtenida hasta el momento. Finalmente, la Heurística Estabilidad de Salones es ejecutada nuevamente.

```
Entrada: Mejor solución de la Fase I, Parámetros
Crear lista tabú vacía
while no se alcance el número de iteraciones definido
    Definir los vecindarios usando N1 y N2
    for i=0: % de los vecindarios
        Generar una solución vecina i
        Calcular la aptitud de la solución generada
    end
    if existe una solución vecina que supere a la mejor solución actual
       la solución vecina sustituye a la solución actual
       Registrar los eventos involucrados en el movimiento como tabús durante
       cierto número de iteraciones
    else
        la mejor solución vecina no tabú, es registrada como la solución actual
        Registrar los eventos involucrados en el movimiento como tabús durante
        cierto número de iteraciones
    end
end
Aplicar la Heurística de Estabilidad de salones sobre la mejor solución obtenida de TS
Salida: Solución cercana a la óptima
```

Figura 21. Algoritmo de Búsqueda Tabú

5. Experimentación

El algoritmo para la solución del UCTP se programó en la herramienta Matlab® en su versión R2107a (Apéndice C), mientras que, el diseño factorial se desarrolló en el software estadístico Minitab® 18. Ambos ejecutados en un equipo con un procesador Intel Core i7 2700 @3.6 GHz y 8 Gb de RAM.

Las instancias a tratar son las generadas a partir de la programación de asignaturas de los periodos académicos 2018-1 y 2018-2 del programa de Ingeniería Industrial de la UIS. Dichas instancias están consignadas en los apéndices D y E, respectivamente.

5.1 Diseño factorial

El algoritmo HGATS cuenta con siete parámetros que guían su desempeño, los cuales son: tamaño de la población, cantidad de generaciones, probabilidad de mutación, número de pasos, porcentaje de vecindario, tamaño de la lista Tabú y cantidad de iteraciones. Con el objetivo de fijar los parámetros se diseñó un diseño factorial fraccionado 2^{k-3} con 5 réplicas. Los detalles relacionados al diseño de cada instancia, están documentados en los apéndices F y G.

Los factores y niveles considerados se presentan en la tabla 10.

Tabla 12.
Factores y niveles del diseño experimental

Factores	**id**	**Nivel bajo (-)**	**Nivel alto (+)**
Tamaño de la población	pob	50	100
Cantidad de generaciones	gen	20	40
Probabilidad de mutación	mut	0,1	0,5
Número de pasos	pasos	50	100
Porcentaje de vecindario	vec	0,01	0,05
Tamaño de la lista Tabú	tabu	5	10
Cantidad de iteraciones	ite	20	40

A continuación, se presenta el Análisis de varianza de cada de las instancias definidas.

5.1.1 **Análisis de varianza para la instancia 2018-1**

Tabla 13.
Análisis de varianza para la instancia 2018-1

Fuente	**GL**	**SC Ajust.**	**MC Ajust.**	**Valor F**	**Valor p**
Modelo	15	85,614	5,7076	1,50	0,132
Lineal	7	54,405	7,7721	2,04	0,063
pob	1	1,661	1,6613	0,44	0,511
gen	1	16,679	16,6792	4,38	0,040
mut	1	0,033	0,0326	0,01	0,927
pasos	1	4,533	4,5329	1,19	0,279
vec	1	26,895	26,8954	7,07	0,010
tabu	1	0,687	0,6871	0,18	0,672
ite	1	3,916	3,9161	1,03	0,314
Interacciones de 2 términos	7	23,880	3,4114	0,90	0,515
pob*gen	1	1,121	1,1213	0,29	0,589
pob*mut	1	0,501	0,5006	0,13	0,718
pob*pasos	1	4,398	4,3979	1,16	0,286
pob*vec	1	0,018	0,0184	0,00	0,945
pob*tabu	1	11,847	11,8470	3,11	0,082
pob*ite	1	4,533	4,5329	1,19	0,279
gen*pasos	1	1,462	1,4619	0,38	0,538
Interacciones de 3 términos	1	7,329	7,3291	1,93	0,170
pob*gen*pasos	1	7,329	7,3291	1,93	0,170
Error	64	243,637	3,8068		
Total	79	329,251			

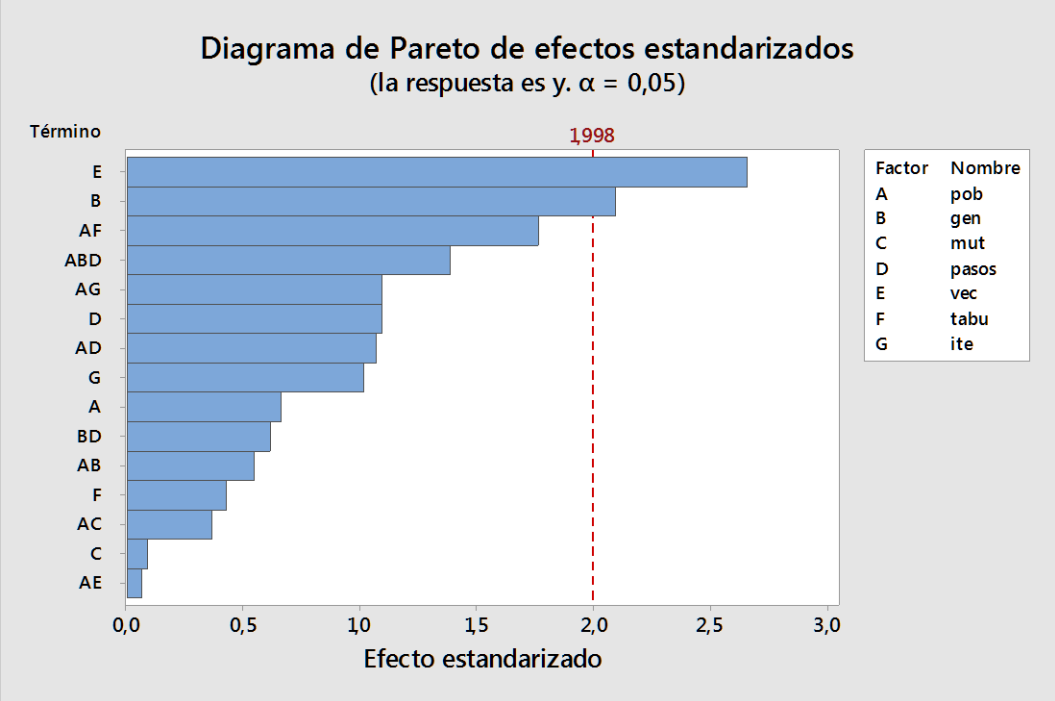

Figura 22. Diagrama de Pareto de efectos estandarizados de la instancia 2018-1

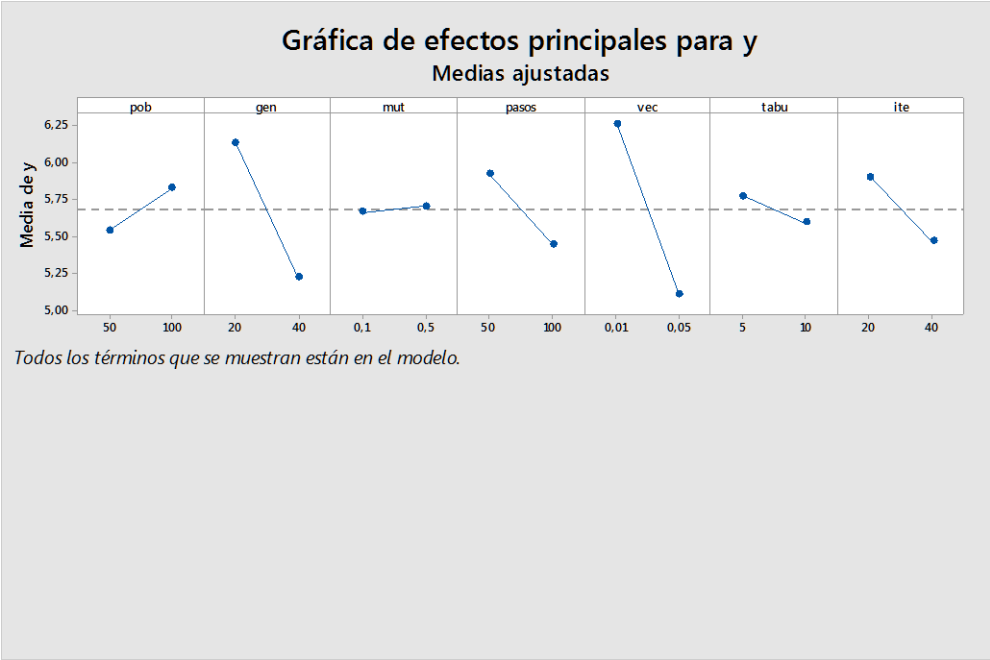

Figura 23.Diagrama de efectos principales para la instancia 2018-1

El análisis de varianza y el diagrama de Pareto de la Figura 21 permiten identificar los factores con efectos significativos sobre la aptitud de la solución, los cuales son: la cantidad de generaciones y el porcentaje del vecindario con valores p de 0.04 y 0.01, respectivamente.

De la gráfica de efectos principales se obtiene que para el objetivo de minimización los parámetros de cantidad de generaciones y porcentaje de vecindario deben tomar los valores de 40 y 0.05. Los demás parámetros, al no ser significativos se fijan en el nivel bajo correspondiente con el fin de reducir el tiempo computacional exigido.

5.1.2 **Análisis de varianza para la instancia 2018-2**

Tabla 14.
Análisis de varianza para la instancia 2018-2

Fuente	**GL**	**SC Ajust.**	**MC Ajust.**	**Valor F**	**Valor p**
Modelo	15	175,389	11,6926	1,73	0,068
Lineal	7	111,589	15,9412	2,35	0,033
pob	1	1,352	1,3520	0,20	0,657
gen	1	7,303	7,3032	1,08	0,303
mut	1	3,433	3,4327	0,51	0,479
pasos	1	35,188	35,1883	5,19	0,026
vec	1	50,653	50,6529	7,48	0,008
tabu	1	1,065	1,0646	0,16	0,693
ite	1	12,595	12,5951	1,86	0,178
Interacciones de 2 términos	7	46,740	6,6772	0,99	0,450
pob*gen	1	2,932	2,9316	0,43	0,513
pob*mut	1	2,178	2,1780	0,32	0,573
pob*pasos	1	0,327	0,3269	0,05	0,827
pob*vec	1	1,552	1,5520	0,23	0,634
pob*tabu	1	32,077	32,0768	4,73	0,033
pob*ite	1	3,053	3,0532	0,45	0,504
gen*pasos	1	4,622	4,6217	0,68	0,412
Interacciones de 3 términos	1	17,060	17,0597	2,52	0,117
pob*gen*pasos	1	17,060	17,0597	2,52	0,117
Error	64	433,595	6,7749		
Total	79	608,984			

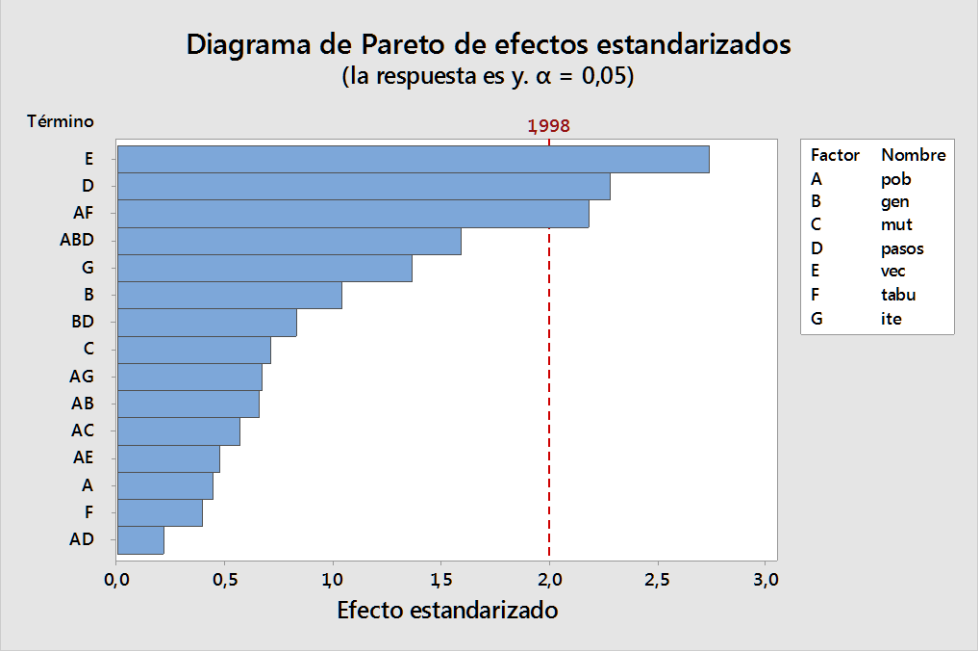

Figura 24. Diagrama de Pareto de efectos estandarizados de la instancia 2018-2

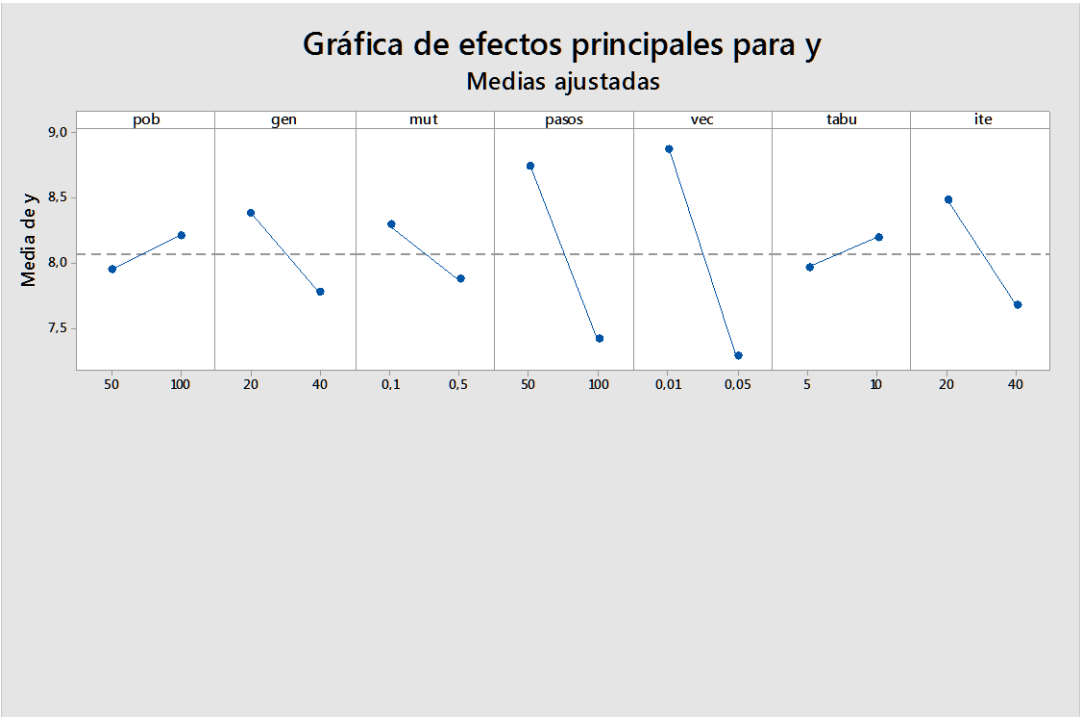

Figura 25. Diagrama de efectos principales para la instancia 2018-2

El análisis de varianza y el diagrama de Pareto de la Figura 23 permiten identificar los factores con efectos significativos sobre la aptitud de la solución, los cuales son: el número de pasos y el porcentaje del vecindario con valores p de 0.026 y 0.008, respectivamente. Además, se observa que existe una interacción significativa entre el tamaño de la población y el tamaño dela lista tabú, con valor p de 0,033.

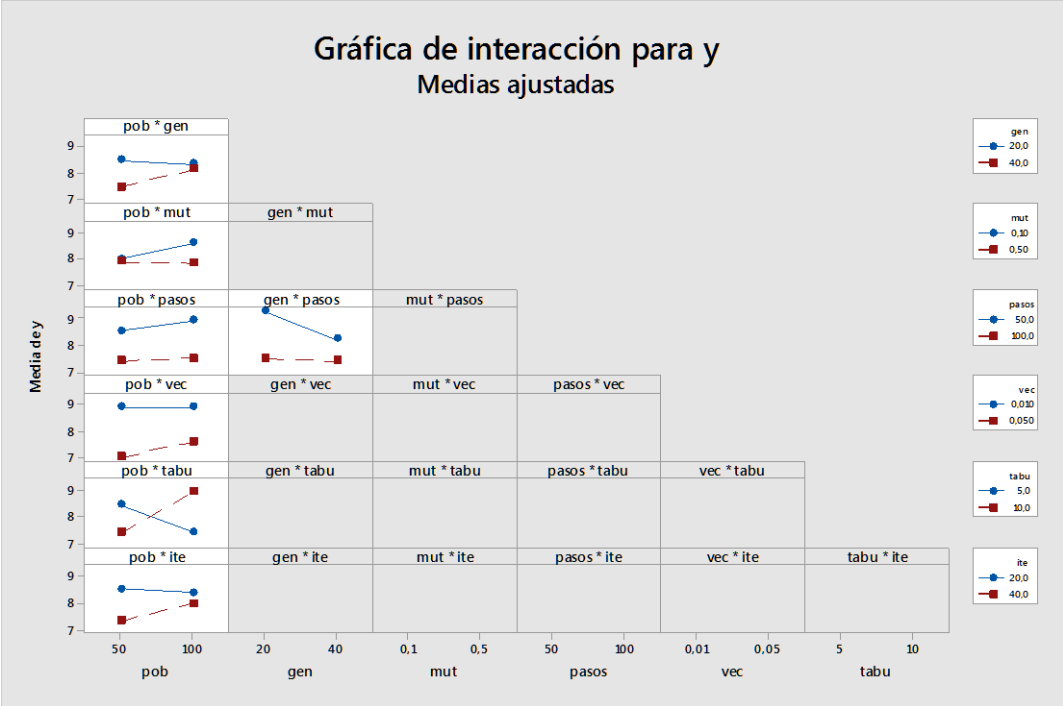

Figura 26. Gráfica de interacciones para la instancia 2018-2

De las gráficas de efectos principales y de interacciones se obtiene que, para el objetivo de minimización los parámetros de número de pasos y porcentaje de vecindario deben tomar los valores de 100 y 0.05, mientras que, los parámetros de tamaño de la población y tamaño de la lista tabú, toman valores de 50 y 10, respectivamente. Los demás parámetros, al no ser significativos se fijan en el nivel bajo correspondiente con el fin de reducir el tiempo computacional.

A continuación, se resumen los valores fijados de los parámetros para cada una de las instancias.

Tabla 15.
Definición de parámetros para las instancias 2018-1 y 2018-2

Instancia	pob	gen	mut	pasos	vec	tabu	ite
2018-1	50	40	0.1	50	0.05	5	20
2018-2	50	20	0,1	100	0.05	10	20

Al realizar el ajuste parámetros usando un diseño factorial fraccionado, debe tenerse en cuenta que, aunque este diseño permite evaluar el desempeño del algoritmo con pocos tratamientos, es posible caer en errores por la confusión entre factores e interacciones debido a la baja resolución del diseño.

5.2 Evaluación de resultados

En la tabla 15 se presentan los resultados de la mejor solución obtenida luego de 10 corridas del algoritmo HGATS Adaptado y el tiempo computacional empleado en cada instancia. El valor obtenido (menor de 1000) en la aptitud de la solución, demuestra que, la solución obtenida es factible. Por otra parte, la tabla 16 presenta detalladamente los valores de penalización asociados a cada restricción descrita en la función de aptitud de la sección 4.3.

Tabla 16.
Resultados obtenidos por el algoritmo HGATS Adaptado

Instancia	Aptitud	Tiempo (s)
2018-1	4,01428571	1089,94856
2018-2	3,25714286	652,307742

Tabla 17.
Valoración por restricciones de la función de aptitud

Instancias	Restricciones duras							Restricciones suaves			
	R3	**R4**	**R6**	**R7**	**R13**	**R15**	**R18**	**R1**	**R14**	**R16**	**R17**
2018-1	0	0	0	0	0	0	0	3,0142	1	0	0
2018-2	0	0	0	0	0	0	0	3,2571	0	0	0

Respecto al tiempo computacional empleado, puede catalogarse como aceptable considerando que, el tiempo dedicado exclusivamente a realizar la programación de asignaturas suele tomar entre 3 y 4 horas, según palabras del Coordinador Académico.

Las restricciones consideradas dentro del problema hacen parte de la realidad actual del proceso de programación de asignaturas en EEIE, sin embargo, no es posible llevar a cabo una comparación directa entre el algoritmo propuesto y la programación manual realizada para las instancias en cuestión, debido a que el Coordinador Académico encargado, en su momento, no necesariamente utilizó dichos criterios al momento de realizar la programación. Considerando lo anteriormente descrito, es interesante observar el resultado obtenido en ambas técnicas usando el Uso Porcentual Semanal (UPS) de los salones, aplicado por Abdelhalim y Khayat (2016) como indicador, sin pretender realizar una comparación directa entre ellas. Las tablas 17 y 18 resumen la información de dicho indicador para cada una de las instancias a tratar.

El indicador UPS representa el porcentaje de franjas en las que un salón está ocupado respecto al total de franjas en las que está disponible para ser programado durante una semana.

Tabla 18.
Indicador UPS para la instancia 2018-1

Salón	Tipo de sala	UPS HGATS	UPS manual
101	Aula de clase	50,0%	58,6%
103	Aula de clase	50,0%	51,4%
118	Aula de clase	40,0%	44,3%
201	Aula de clase	51,4%	60,0%
206	Aula de clase	57,1%	48,6%
301	Aula de clase	45,7%	42,9%
302	Aula de clase	42,9%	34,3%
303	Aula de clase	31,4%	22,9%
304	Aula de clase	51,4%	35,7%
305	Aula de clase	37,1%	55,7%
306	Aula de clase	40,0%	45,7%
307	Aula de clase	27,1%	35,7%
309	Aula de clase	34,3%	27,1%
310	Aula de clase	22,9%	34,3%
402	Aula de clase	22,9%	20,0%
403	Aula de clase	17,1%	15,7%
111	Sala de computo	48,6%	37,1%
317	Laboratorio de calidad	17,1%	17,1%
UPS Promedio		**38,1%**	

Tabla 19.
Indicador UPS para la instancia 2018-2

Salón	Tipo de sala	UPS HGATS	UPS manual
101	Aula de clase	67,1%	40,0%
103	Aula de clase	48,6%	34,3%
118	Aula de clase	48,6%	52,9%
201	Aula de clase	40,0%	48,6%
206	Aula de clase	32,9%	51,4%
301	Aula de clase	48,6%	31,4%
302	Aula de clase	48,6%	45,7%
303	Aula de clase	34,3%	62,9%
304	Aula de clase	51,4%	61,4%
305	Aula de clase	42,9%	50,0%
306	Aula de clase	28,6%	44,3%
307	Aula de clase	48,6%	35,7%
309	Aula de clase	40,0%	34,3%
310	Aula de clase	34,3%	40,0%
402	Aula de clase	40,0%	31,4%
403	Aula de clase	20,0%	18,6%
111	Sala de computo	40,0%	31,4%
317	Laboratorio de calidad	28,6%	28,6%
UPS Promedio		**39,7%**	

6. Conclusiones

Como pudo observarse en la Revisión de literatura, la mayoría de trabajos e investigaciones encontradas suelen enfocarse en la evaluación de algoritmos sobre instancias de referencia, por lo que la aplicación de estos algoritmos en instancias del mundo real, como lo es este proyecto, sirve como referencia a investigaciones futuras que estén enfocadas en el uso de metaheurísticas para resolver el UCTP en instituciones educativas con condiciones similares.

Se ha documentado un Modelo de Programación Lineal Entera Mixta en el que se describen cada una de las restricciones del problema en EEIE, y se identifican cerca de 9 millones de variables y 20 millones de ecuaciones, lo cual representa una cantidad significativa de esfuerzo computacional al momento de buscar una solución óptima al problema.

La versatilidad de las técnicas metaheurísticas han ampliado la posibilidad de encontrar buenas soluciones a problemas de alto nivel de complejidad. Entre estas, el Algoritmo Genético y el Algoritmo de Búsqueda Tabú han sido ampliamente documentados en el área de los horarios universitarios, lo cual facilita extender su aplicación a problemas específicos, como nuestro caso de estudio.

Dado que, las programaciones obtenidas eran factibles en todas las corridas realizadas (Apéndice H), el algoritmo HGATS ha demostrado ser efectivo para el problema propuesto. Además, el tiempo computacional máximo empleado no ha superado los 25 minutos, el cual, es considerado un tiempo razonable.

La definición de parámetros se realizó considerando un Diseño Factorial Fraccionado de resolución IV, el cual genera confusión entre los factores e interacciones, sin embargo, con los parámetros utilizados se obtuvo buenos resultados (factibles) en las soluciones finales.

Se definió el indicador UPS con el fin de observar la forma en que se realiza la asignación de salones dentro de la programación realizada por HGATS. De dicho indicador, se observa que, para la instancia 2018-1 solo se usan el 38,1% de los espacios disponibles, mientras que, para la instancia 2018-2 este valor es del 39,7%. Estos valores, indican que los salones no están siendo usados eficientemente.

7. Recomendaciones

Ampliar el alcance del problema mediante la inclusión de los datos de matrículas de los estudiantes, la capacidad de los salones y los elementos excluidos en la sección 3.1.3, con el fin de evaluar el impacto que tendrían dichas modificaciones en la Programación de Asignaturas.

Programar el modelo PLEM definido en el capítulo 3, usando GAMS, con el fin de obtener la solución óptima del problema y comparar el desempeño de la metaheurística HGATS respecto a dicha solución.

Debido al bajo UPS promedio en ambas instancias se recomienda reducir el número de Aulas de clase disponible para programación. De esta forma, se obliga al algoritmo a buscar buenas soluciones, aprovechando de mejor manera los salones disponibles y consecuentemente, mejorando dicho indicador.

Dado que el algoritmo híbrido HGATS trabaja en fases separadas, es conveniente evaluar el desempeño del Algoritmo Genético y Búsqueda Tabú independientemente, con el objetivo de explotar adecuadamente las capacidades de cada uno en tiempos de computo razonables.

Implementar otras metaheurísticas prometedoras documentadas en la literatura, como Recocido Simulado y sus variantes, con el fin de verificar su desempeño en el problema de programación de cursos universitarios descrito para EEIE.

Referencias bibliográficas

Abdelhalim, E. A., y Khayat, G. A. El. (2016). A Utilization-based Genetic Algorithm for Solving the University Timetabling Problem. *Alexandria Engineering Journal*, *55*(2), 1395-1409. https://doi.org/10.1016/j.aej.2016.02.017

Abdullah, S. (2006). *Heuristic Approaches for University Timetabling Problems (Tesis doctoral)*. University of Nottingham. Recuperado a partir de https://goo.gl/a41Qdo

Abdullah, S., Burke, E. K., y Mccollum, B. (2007). Using a randomised iterative improvement algorithm with composite neighbourhood structures for the university course timetabling problem. *Metaheuristics*, 153-169. https://doi.org/10.1007/978-0-387-71921-4

Abdullah, S., Shaker, K., y Shaker, H. (2011). Investigating a round robin strategy over multi algorithms in optimising the quality of university course timetables. *Int. J. Phys. Sci*, *6*(6), 1452-1462. https://doi.org/10.5897/IJPS11.210

Abdullah, S., Turabieh, H., McCollum, B., y McMullan, P. (2012). A hybrid metaheuristic approach to the university course timetabling problem. *Journal of Heuristics*, *18*(1), 1-23. https://doi.org/10.1007/s10732-010-9154-y

Abuhamdah, A. (2015). Adaptive Acceptance Criterion (AAC) Algorithm for Optimization Problems. https://doi.org/10.3844/jcssp.2015.675.691

Abuhamdah, A., y Ayob, M. (2010). Adaptive randomized descent algorithm for solving course timetabling problems, *5*(16), 2516-2522.

Abuhamdah, A., Ayob, M., Kendall, G., y Sabar, N. R. (2014). Population based Local Search for university course timetabling problems. *Appl Intell*, 44-53. https://doi.org/10.1007/s10489-013-0444-6

Al-betar, M. A., y Khader, A. T. (2010). A harmony search algorithm for university course

timetabling. *Ann Oper Res*, *194*, 3-31. https://doi.org/10.1007/s10479-010-0769-z

Al-betar, M. A., Khader, A. T., y Liao, I. Y. (2010). A Harmony Search with Multi-pitch Adjusting Rate for the University Course Timetabling, 147-161.

Al-Betar, M. A., Khader, A. T., y Zaman, M. (2012). University course timetabling using a hybrid harmony search metaheuristic algorithm. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, *42*(5), 664-681. https://doi.org/10.1109/TSMCC.2011.2174356

Angeles, A. N. (2015). *ELABORACIÓN DE UNA SOLUCIÓN METAHEURÍSTICA USANDO UN ALGORITMO GENÉTICO QUE PERMITA ELABORAR LA DISTRIBUCIÓN DE LOS HORARIOS ACADÉMICOS (Tesis de pregrado)*. Pontificia Universidad Católica del Perú. Recuperado a partir de http://tesis.pucp.edu.pe/repositorio/handle/123456789/6057

Avella, P., y Vasil'ev, I. (2005). a Computational Study of a Cutting Plane Algorithm for University Course Timetabling. *Journal of Scheduling*, *2*, 497-514. https://doi.org/https://doi.org/10.1007/s10951-005-4780-1

Badoni, R. P., Gupta, D. K., y Mishra, P. (2014). A new hybrid algorithm for university course timetabling problem using events based on groupings of students. *COMPUTERS & INDUSTRIAL ENGINEERING*, *78*, 12-25. https://doi.org/10.1016/j.cie.2014.09.020

Blaz Aristo, S. P. (2016). *Un sistema de generación de horarios para la enseñanza de pregrado en universidades peruanas mediante algoritmos genéticos (Tesis de pregrado)*. *Repositorio de Tesis - UNMSM*. UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS. Recuperado a partir de http://cybertesis.unmsm.edu.pe/handle/cybertesis/4943

Blum, C., y Roli, A. (2003). Metaheuristics in combinatorial optimization: overview and conceptual comparison. *ACM Computing Surveys*, *35*(3), 189-213.

https://doi.org/10.1007/s10479-005-3971-7

Bolaji, A. L., Khader, A. T., Al-Betar, M. A., y Awadallah, M. A. (2011). Artificial Bee Colony Algorithm for Curriculum-Based Course Timetabling Problem. En *ICIT 2011 The 5th International Conference on Information Technology*. Recuperado a partir de https://www.researchgate.net/publication/279447574

Bucco, G. B., y Bandeira, D. L. (2017). Development of a linear programming model for the University Course Timetabling Problem, (1999), 40-49.

Burke, E. K., y Bykov, Y. (2008). A late acceptance strategy in hill-climbing for exam timetabling problems. *PATAT 2008 Conference, Montreal, Canada*, (January 2008), 1-7. Recuperado a partir de http://patatconference.org/patat2008/proceedings/Bykov-HC2a.pdf%0Ahttp://www.cs.nott.ac.uk/~yxb/LAHC/LAHC-PATAT.pdf

Burke, E. K., Mareček, J., Parkes, A. J., y Rudová, H. (2012). A branch-and-cut procedure for the Udine Course Timetabling problem. *Annals of Operations Research*, *194*(1), 71-87. https://doi.org/10.1007/s10479-010-0828-5

Chavez, O., Pozos, P., y Lengyel, F. (2011). Solving the International Timetabling Competition : a Deterministic Approach, *113*, 1-18. https://doi.org/10.3233/FI-2011-596

Chen, R.-M., y Shih, H.-F. (2013). Solving University Course Timetabling Problems Using Constriction Particle Swarm Optimization with Local Search. *Algorithms*, *6*(2), 227-244. https://doi.org/10.3390/a6020227

Chinnasri, W., y Sureerattanan, N. (2012). Performance Study of Genetic Operators on University Course Timetabling Problem, *4*(March). https://doi.org/10.4156/ijact.vol4.issue20.8

Clausen, J. (1999). Branch and bound algorithms-principles and examples. *Department of Computer Science, University of …*, 1-30. Recuperado a partir de

http://www.imada.sdu.dk/~jbj/heuristikker/TSPtext.pdf

Cook, W. J., Cunningham, W. H., Pulleyblank, W. R., y Schrijver, A. (1998). Combinatorial optimization. *Computers & Mathematics with Applications*, *35*(9), 139. https://doi.org/10.1016/S0898-1221(98)90683-6

Cruz, M. A., Flores, M., Martínez, A., Moreno, P., y Cruz, M. H. (2016). Solving a Real Constraint Satisfaction Model for the University Course Timetabling Problem : A Case Study. *Hindawi Publishing Corporation*. https://doi.org/10.1155/2016/7194864

Herrera, F. (2009). Introducción a los Algoritmos Metaheurísticos.

Hillier, F., y Lieberman, G. (2010). *Introduccion a la investigacion de operaciones* (McGraw-Hil). Ciudad de México, México.

Jat, S. N., y Yang, S. (2011). A hybrid genetic algorithm and tabu search approach for post enrolment course timetabling, 617-637. https://doi.org/10.1007/s10951-010-0202-0

Kohshori, M. S., y Abadeh, M. S. (2012). Hybrid Genetic Algorithms for University Course Timetabling, *9*(2), 446-455.

Lach, G., y Lübbecke, M. E. (2012). Curriculum based course timetabling: New solutions to Udine benchmark instances. *Annals of Operations Research*, *194*(1), 255-272. https://doi.org/10.1007/s10479-010-0700-7

Lewis, R. (2007). *Metaheuristics for University Course Timetabling (Tesis doctoral)*. *Studies in Computational Intelligence*. https://doi.org/10.1007/978-3-540-48584-1

Mushi, A. R. (2006). Tabu search heuristic for university course timetabling problem. *African Journal of Science and Technology*, *7*(1), 34-40. https://doi.org/http://dx.doi.org/10.4314/ajst.v7i1.55191

Nagata, Y., y Ono, I. (2014). Random Partial Neighborhood Search for University Course

Timetabling Problem. *Parallel Problem Solving from Nature -- PPSN XIII*, *8672*, 782-791. https://doi.org/10.1007/978-3-319-10762-2_77

Obit, J. H. (2010). *Developing novel meta-heuristic , hyper-heuristic and cooperative search for course timetabling problems (Tesis doctoral)*. University of Nottingham. Recuperado a partir de http://eprints.nottingham.ac.uk/13581/1/537629.pdf

Obit, J. H., y Landa-silva, D. (2010). Computational Study of Non-linear Great Deluge for University Course Timetabling, 309-328.

Ortiz, C. (2010). *Algoritmos heurísticos y metaheurísticos para el problema de localización de regeneradores. (Tesis de pregrado)*. Universidad Rey Juan Carlos. Recuperado a partir de http://eciencia.urjc.es/handle/10115/4129

Pongcharoen, P., Promtet, W., Yenradee, P., y Hicks, C. (2008). Stochastic Optimisation Timetabling Tool for university course scheduling, *112*, 903-918. https://doi.org/10.1016/j.ijpe.2007.07.009

Rossi-doria, O., Sampels, M., Birattari, M., Chiarandini, M., Dorigo, M., Gambardella, L. M., y Manfrin, M. (s. f.). A Comparison of the Performance of Different Metaheuristics on the Timetabling Problem, 329-351.

Sanchis, A., Ledezma, A., Iglesias, J., García, B., y Alosnso, J. (s. f.). *Complejidad Computacional*. Recuperado a partir de http://ocw.uc3m.es/ingenieria-informatica/teoria-de-automatas-y-lenguajes-formales/material-de-clase-1/tema-8-complejidad-computacional

Schaerf, A. (1999). A Survey of Automated Timetabling. *Artificial Intelligence Review*, *13(2)*, 87-127.

Schimmelpfeng, K., y Helber, S. (2007). Application of a real-world university-course timetabling model solved by integer programming. *OR Spectrum*, *29*(4), 783-803.

https://doi.org/10.1007/s00291-006-0074-z

Taha, H. (2012). *Investigación de operaciones* (Pearson). Ciudad de México, México.

Tarawneh, H. Y., & Ayob, M. (2013). Adaptative neighbourhoods structure selection mechanism in simulated annealing for solving university course timetabling problems.

Tarawneh, H. Y., Ayob, M., y Ahmad, Z. (2013). A hybrid simulated annealing with solutions memory for curriculum-based course timetabling problem. *Journal of Applied Sciences*. https://doi.org/10.3923/jas.2013.262.269

Teoh, C., Haron, H., y Wibowo, A. (2016). Integrating a Repairing-Based Genetic Algorithm-Neighborhood Search Structure in Solving the Course Timetabling Problem, *2002*. https://doi.org/10.3844/jcssp.2016.510.516

Vermuyten, H., Lemmens, S., Marques, I., y Beliën, J. (2016). Developing compact course timetables with optimized student flows. *European Journal of Operational Research*, *251*(2), 651-661. https://doi.org/10.1016/j.ejor.2015.11.028

Wahid, J., y Hussin, N. M. (2016). Construction of Initial Solution Population for Curriculum-Based Course Timetabling using Combination of Graph Heuristics, *8*(8), 91-95.

Wahid, J., y Hussin, N. M. (2017). Hybrid harmony search with great deluge for UUM CAS curriculum based course timetabling. *Journal of Telecommunication, Electronic and Computer Engineering*, *9*(1-2), 33-38. Recuperado a partir de https://www.scopus.com/inward/record.uri?eid=2-s2.0-85020760855&partnerID=40&md5=778f9938403a71fcbc022a0be4bd6fb3

Wibowo, A. (2014). An Adapted Cuckoo Optimization Algorithm and Genetic Algorithm Approach to the University Course Timetabling problem. *International Journal of Computational Intelligence and Applications*, *13 (1)*, 13.

https://doi.org/10.1142/S1469026814500023

Yang, S., y Jat, S. N. (2011). Genetic Algorithms With Guided and Local Search Strategies for University Course Timetabling, *41*(1), 93-106.

Zheng, S., Wang, L., Liu, Y., y Zhang, R. (2015). A simulated annealing algorithm for university course timetabling considering travelling distances. *International Journal of Computing Science and Mathematics*, *6*(2), 139. https://doi.org/10.1504/IJCSM.2015.069461