UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

FACULTAD DE CONTADURÍA, ADMINISTRACIÓN E INFORMÁTICA

MAESTRÍA EN OPTIMIZACIÓN Y CÓMPUTO APLICADO

Programación de horarios universitarios jerárquicos 2019

TESIS PROFESIONAL

Para obtener el Grado de Maestría en Optimización y Cómputo Aplicado

Presenta

Eduardo Flores Gomez

Director de Tesis

Dr. Federico Alonso Pecina

Co-Director

Dr. David Romero Vargas †

Revisores:

Dra. Irma Yazmín Hernández Báez

Dr. Martín Heriberto Cruz Rosales

Dr. Marco Antonio Cruz Chávez

Dr. José Crispín Zavala Díaz

CUERNAVACA, MORELOS MARZO,2021

Cuernavaca, Morelos a 15 de enero del 2021.

DR. AUGUSTO RENATO PÉREZ MAYO
SECRETARIO DE INVESTIGACIÓN Y POSGRADO DE LA FCAeI
PRESENTE

En mi carácter de revisor de Tesis, hago de su conocimiento que he leído con interés la tesis para obtener el grado de la Maestría Optimización y Cómputo Aplicado, del estudiante Eduardo Flores Gomez, con matricula 10023091, con el título **Programación de horarios universitarios jerárquicos 2019** por lo cual, me permito informarle que después de una revisión cuidadosa de dicha tesis, concluyo que el trabajo se caracteriza por el establecimiento de objetivos académicos pertinentes y una metodología adecuada para su logro. Además construye una estructura coherente y bien documentada, por lo cual considero que los resultados obtenidos contribuyen al conocimiento del tema tratado.

Con base en los argumentos precedentes me permito expresar mi **VOTO APROBATORIO** por lo que de mi parte no existe inconveniente para que el estudiante continúe con los trámites que esta Secretaría de Investigación y Posgrado tenga establecidos para obtener el grado mencionado.

Atentamente
Por una humanidad culta
Una universidad de excelencia

Dr. Federico Alonso Pecina
Profesor- investigador
Facultad de Contaduría, Administración e Informática

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

Se expide el presente documento firmado electrónicamente de conformidad con el ACUERDO GENERAL PARA LA CONTINUIDAD DEL FUNCIONAMIENTO DE LA UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS DURANTE LA EMERGENCIA SANITARIA PROVOCADA POR EL VIRUS SARS-COV2 (COVID-19) emitido el 27 de abril del 2020.

El presente documento cuenta con la firma electrónica UAEM del funcionario universitario competente, amparada por un certificado vigente a la fecha de su elaboración y es válido de conformidad con los LINEAMIENTOS EN MATERIA DE FIRMA ELECTRÓNICA PARA LA UNIVERSIDAD AUTÓNOMA DE ESTADO DE MORELOS emitidos el 13 de noviembre del 2019 mediante circular No. 32.

Sello electrónico

FEDERICO ALONSO PECINA | Fecha:2021-03-23 10:00:43 | Firmante

f7GUrureAnDUbJA7P7w9OBzGT0PvcS8jvwOZM7x34qNd9NPG7/yctC29U8hhaXqiVLRKXP45nmIZfLEBUQDOAPRmnr132fsdSFyy6NhtxjQlUrSrjHnEYj9irKhlvhsFeywJG9+cPWx7Hx86M1v4BMFfnK0kxkPdwEAmjPCp2KSaQKtDRPHXdTRThUzcvmr7dyIGTupziYFe4OdvGXgxnjM8Xoiw/eukxMd+pE5I4w17agB7FSWErwCUujbf6hUJ4LsY0intc0OpcoSMF+G0HAMz/S8lJrc/kOHOKikWnoPb/5UWlvmf+re8Q+/XLhOalCnSrqYLlv82J/n/vTZtoQ==

Puede verificar la autenticidad del documento en la siguiente dirección electrónica o escaneando el código QR ingresando la siguiente clave:

adSRU5

https://efirma.uaem.mx/noRepudio/LnCLjLjWilOXIgCPOBARFC0PXpFCa7Lx

Cuernavaca, Morelos a 15 de enero del 2021.

DR. AUGUSTO RENATO PÉREZ MAYO
SECRETARIO DE INVESTIGACIÓN Y POSGRADO DE LA FCAeI
PRESENTE

En mi carácter de revisor de Tesis, hago de su conocimiento que he leído con interés la tesis para obtener el grado de la Maestría Optimización y Cómputo Aplicado, del estudiante Eduardo Flores Gomez, con matricula 10023091, con el título **Programación de horarios universitarios jerárquicos 2019** por lo cual, me permito informarle que después de una revisión cuidadosa de dicha tesis, concluyo que el trabajo se caracteriza por el establecimiento de objetivos académicos pertinentes y una metodología adecuada para su logro. Además construye una estructura coherente y bien documentada, por lo cual considero que los resultados obtenidos contribuyen al conocimiento del tema tratado.

Con base en los argumentos precedentes me permito expresar mi **VOTO APROBATORIO** por lo que de mi parte no existe inconveniente para que el estudiante continúe con los trámites que esta Secretaría de Investigación y Posgrado tenga establecidos para obtener el grado mencionado.

Atentamente
Por una humanidad culta
Una universidad de excelencia

Dra. Irma Yazmín Hernández Báez
Profesor- investigador
Universidad Politécnica del Estado de Morelos

Se expide el presente documento firmado electrónicamente de conformidad con el ACUERDO GENERAL PARA LA CONTINUIDAD DEL FUNCIONAMIENTO DE LA UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS DURANTE LA EMERGENCIA SANITARIA PROVOCADA POR EL VIRUS SARS-COV2 (COVID-19) emitido el 27 de abril del 2020.

El presente documento cuenta con la firma electrónica UAEM del funcionario universitario competente, amparada por un certificado vigente a la fecha de su elaboración y es válido de conformidad con los LINEAMIENTOS EN MATERIA DE FIRMA ELECTRÓNICA PARA LA UNIVERSIDAD AUTÓNOMA DE ESTADO DE MORELOS emitidos el 13 de noviembre del 2019 mediante circular No. 32.

Sello electrónico

IRMA YAZMIN HERNANDEZ BAEZ | Fecha:2021-01-25 15:57:20 | Firmante

wBd6WxQKGwNJiLdqWiFE0EGkftV2L1JQoXAcX4avh3KD1cWD9YEPpr0ZYBEJ+pmpdcQn/j+ydffEr6gAtKEoJFuyH1plGceObVe8qdUbsuqcrE4s3lp087qd6OKV2HtZwLzpkmHBH2s9M77h2pwwXYJX3rXCZUCnPnUg1Niz/0kQkQv85quk+psUQBdprxDM+9MCO9TbQdYYnFtA2iXOv1XIo0EHG7gk+alifJ41Sl+ay+1qEWe4KHVY0vOMgWWl8To2+7TuFVd0KKscdIa4NPL8RhPp+vcsHdKlJsGKL35IRNwTJAJItktouHg9xtKuzalpZHPFTkBWKOSw6q2msg==

Puede verificar la autenticidad del documento en la siguiente dirección electrónica o escaneando el código QR ingresando la siguiente clave:

EHoxRl

https://efirma.uaem.mx/noRepudio/uXMHNqXwblDQhldzhW14PT2o2kdGEaxL

Cuernavaca, Morelos a 15 de enero del 2021.

DR. AUGUSTO RENATO PÉREZ MAYO
SECRETARIO DE INVESTIGACIÓN Y POSGRADO DE LA FCAeI
PRESENTE

En mi carácter de revisor de Tesis, hago de su conocimiento que he leído con interés la tesis para obtener el grado de la Maestría Optimización y Cómputo Aplicado, del estudiante Eduardo Flores Gomez, con matricula 10023091, con el título **Programación de horarios universitarios jerárquicos 2019** por lo cual, me permito informarle que después de una revisión cuidadosa de dicha tesis, concluyo que el trabajo se caracteriza por el establecimiento de objetivos académicos pertinentes y una metodología adecuada para su logro. Además construye una estructura coherente y bien documentada, por lo cual considero que los resultados obtenidos contribuyen al conocimiento del tema tratado.

Con base en los argumentos precedentes me permito expresar mi **VOTO APROBATORIO** por lo que de mi parte no existe inconveniente para que el estudiante continúe con los trámites que esta Secretaría de Investigación y Posgrado tenga establecidos para obtener el grado mencionado.

Atentamente
Por una humanidad culta
Una universidad de excelencia

Dr. Martín Heriberto Cruz Rosales
Profesor- investigador
Facultad de Contaduría, Administración e Informática

Se expide el presente documento firmado electrónicamente de conformidad con el ACUERDO GENERAL PARA LA CONTINUIDAD DEL FUNCIONAMIENTO DE LA UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS DURANTE LA EMERGENCIA SANITARIA PROVOCADA POR EL VIRUS SARS-COV2 (COVID-19) emitido el 27 de abril del 2020.

El presente documento cuenta con la firma electrónica UAEM del funcionario universitario competente, amparada por un certificado vigente a la fecha de su elaboración y es válido de conformidad con los LINEAMIENTOS EN MATERIA DE FIRMA ELECTRÓNICA PARA LA UNIVERSIDAD AUTÓNOMA DE ESTADO DE MORELOS emitidos el 13 de noviembre del 2019 mediante circular No. 32.

Sello electrónico

MARTIN HERIBERTO CRUZ ROSALES | Fecha:2021-01-31 18:10:49 | Firmante

orYfzzUTYwng45nRsqQ5TodB1sl7YTmt1GA5jgxVTOl0f7aTeNkvzmYynrKuxsMAZiP3Gi1tHDllS6QmzJ4WX+Pc9REt0q8KoXRn3PBXBSD405BK3EecDmsrWsjtm96vIjlwK1wdwagxoz3yDHt+u88XjwLYaLcD0MCUSAHivlhTB1MeC3T3JJvnQpKcQ+T9THuBcdpSE/XzaA98XlqVhGVPrw/F7QJGVlTcnWSzK+Zg44cLR9HsLOJiX0Da2PUTCo4aXEbbnJzpZWAFJ/qbtJYJlozJNygtBsbr293IowDsm4b2NtEAgiyVZSYT1NRE5iAXe6rDX3Zs6EiKny4zeA==

Puede verificar la autenticidad del documento en la siguiente dirección electrónica o escaneando el código QR ingresando la siguiente clave:

sV5xe7

https://efirma.uaem.mx/noRepudio/tiqSV8XgIWwzvRdgiiSZaHM6VCxcd1YR

Cuernavaca, Morelos a 15 de enero del 2021.

DR. AUGUSTO RENATO PÉREZ MAYO
SECRETARIO DE INVESTIGACIÓN Y POSGRADO DE LA FCAeI
PRESENTE

En mi carácter de revisor de Tesis, hago de su conocimiento que he leído con interés la tesis para obtener el grado de la Maestría Optimización y Cómputo Aplicado, del estudiante Eduardo Flores Gomez, con matricula 10023091, con el título **Programación de horarios universitarios jerárquicos 2019** por lo cual, me permito informarle que después de una revisión cuidadosa de dicha tesis, concluyo que el trabajo se caracteriza por el establecimiento de objetivos académicos pertinentes y una metodología adecuada para su logro. Además construye una estructura coherente y bien documentada, por lo cual considero que los resultados obtenidos contribuyen al conocimiento del tema tratado.

Con base en los argumentos precedentes me permito expresar mi **VOTO APROBATORIO** por lo que de mi parte no existe inconveniente para que el estudiante continúe con los trámites que esta Secretaría de Investigación y Posgrado tenga establecidos para obtener el grado mencionado.

Atentamente
Por una humanidad culta
Una universidad de excelencia

Dr. Marco Antonio Cruz Chávez
Profesor- investigador
Centro de Investigación en Ingeniería y Ciencias Aplicadas (CIICAP)

Se expide el presente documento firmado electrónicamente de conformidad con el ACUERDO GENERAL PARA LA CONTINUIDAD DEL FUNCIONAMIENTO DE LA UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS DURANTE LA EMERGENCIA SANITARIA PROVOCADA POR EL VIRUS SARS-COV2 (COVID-19) emitido el 27 de abril del 2020.

El presente documento cuenta con la firma electrónica UAEM del funcionario universitario competente, amparada por un certificado vigente a la fecha de su elaboración y es válido de conformidad con los LINEAMIENTOS EN MATERIA DE FIRMA ELECTRÓNICA PARA LA UNIVERSIDAD AUTÓNOMA DE ESTADO DE MORELOS emitidos el 13 de noviembre del 2019 mediante circular No. 32.

Sello electrónico

MARCO ANTONIO CRUZ CHAVEZ | Fecha:2021-04-09 13:27:16 | Firmante

Rs5d8ojsBceYCuAXCYJP4Vf4vKXHdi5NeRZgRbGYH7TIrFAX86LxG8E82yof7SigFZWOocwLhbWhbBgmaT5Bj1YZ4aHy65Sb0TTx10FFxGBgZEWT9n25kvAom7WTwWZnzUuyw4p6sNTOYsLrIFcmfSXDJzUyF3lGGr77y+n16FTivSfXCWVRtRp+DRgHTOEqLW7iu4w8GB1aNUf/raT8HTPi/KaPGxfonHqDmuY5e55iLCKB+ywNgFlfea35elqFSaiD6FbkMtwpb303Ao+o9shojic7HF7EZ6stgXwTZagZZy4I6xklldrx3prQuOszprC9rZPo4eZ8YgwlPJ7Yvg==

Puede verificar la autenticidad del documento en la siguiente dirección electrónica o escaneando el código QR ingresando la siguiente clave:

F9htSN

https://efirma.uaem.mx/noRepudio/wUJTmEPb3gXWfbiaiMyq9WkYY8v89GT3

Cuernavaca, Morelos a 15 de enero del 2021.

DR. AUGUSTO RENATO PÉREZ MAYO
SECRETARIO DE INVESTIGACIÓN Y POSGRADO DE LA FCAeI
PRESENTE

En mi carácter de revisor de Tesis, hago de su conocimiento que he leído con interés la tesis para obtener el grado de la Maestría Optimización y Cómputo Aplicado, del estudiante Eduardo Flores Gomez, con matricula 10023091, con el título **Programación de horarios universitarios jerárquicos 2019** por lo cual, me permito informarle que después de una revisión cuidadosa de dicha tesis, concluyo que el trabajo se caracteriza por el establecimiento de objetivos académicos pertinentes y una metodología adecuada para su logro. Además construye una estructura coherente y bien documentada, por lo cual considero que los resultados obtenidos contribuyen al conocimiento del tema tratado.

Con base en los argumentos precedentes me permito expresar mi **VOTO APROBATORIO** por lo que de mi parte no existe inconveniente para que el estudiante continúe con los trámites que esta Secretaría de Investigación y Posgrado tenga establecidos para obtener el grado mencionado.

Atentamente
Por una humanidad culta
Una universidad de excelencia

Dr. José Crispín Zavala Díaz
Profesor- investigador
Facultad de Contaduría, Administración e Informática

Se expide el presente documento firmado electrónicamente de conformidad con el ACUERDO GENERAL PARA LA CONTINUIDAD DEL FUNCIONAMIENTO DE LA UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS DURANTE LA EMERGENCIA SANITARIA PROVOCADA POR EL VIRUS SARS-COV2 (COVID-19) emitido el 27 de abril del 2020.

El presente documento cuenta con la firma electrónica UAEM del funcionario universitario competente, amparada por un certificado vigente a la fecha de su elaboración y es válido de conformidad con los LINEAMIENTOS EN MATERIA DE FIRMA ELECTRÓNICA PARA LA UNIVERSIDAD AUTÓNOMA DE ESTADO DE MORELOS emitidos el 13 de noviembre del 2019 mediante circular No. 32.

Sello electrónico

JOSE CRISPIN ZAVALA DIAZ | Fecha:2021-01-25 15:21:43 | Firmante

Z1/OABK5TeLObhw3UyvA6Hdav/RjdUlhZSLMHLGJrIqzVhw4twmgSgxkDqicgIqMRNpxMSHhX/zhPsaxIDrVhzqNeqWAv/P3px7DQI1RqoASYxcpcz0RQ6CE+YUHfdYn3VZ0UeFph4Zl/TCvEdmzXRpeE+Wj94Jm8QrxTIezVCo/tUzBxhaD5rgr0dhmikMGTyb4HfAAyB3hpgWuLmo+dsetwzejMAjcjXPGTRFrcMNbuLLv9Rdc8k30wxMY5NLCin9SQGBwtCwJZQH++bWWMo33jFONTDe/pnHKxVVqQnmw38bapQykwcd8A+f6btXXb/VrwvuyaIeFqO0XvQmv4A==

Puede verificar la autenticidad del documento en la siguiente dirección electrónica o escaneando el código QR ingresando la siguiente clave:

YfwazK

https://efirma.uaem.mx/noRepudio/eKIgv0bX6vWdfPoWoQHM7S73aX9yxTIp

Agradecimientos

Quiero agradecer a mis padres, que me han apoyado en todo momento, siempre dándome ese soporte para poder continuar con mis estudios. A mi hermano Marco que siempre me motivaba a seguir adelante. A mi hermosa ahijada Lía, que siempre al verla con su sonrisa encantadora me motivaba a no darme por vencido.

Mi abuelita María que siempre estaba al pendiente de mí y se preocupaba porque no me faltara nada. A mis abuelos paternos y mi abuelo materno, que en paz descansen. Y a toda mi familia que siempre me apoyo en los momentos difíciles de la vida.

A mi asesor Federico Alonso, por su paciencia, por el gran aprendizaje que me transmitió, y por la confianza que me dio para poder realizar este trabajo de tesis.

A mi grupo sinodal que me ayudo a corregir mi tesis, y me guiaron para poder hacer un mejor trabajo, y en general a todos los profesores que tuve la suerte de conocerlos durante la maestría.

En memoria de mi co-director de tesis David Romero, por el privilegio de aprender de su sabiduría. A mis familiares que lamentablemente fallecieron durante la pandemia, pero siempre llevare en mi memoria.

Y principalmente gracias a dios, por brindarme esta oportunidad y no dejarme caer. La vida humana es frágil como la flor. Hoy es, y mañana no existe más. Por tanto, pon a Dios en el fundamento de tus proyectos.

Resumen

En la presente tesis de investigación se aborda el problema de *Timetabling* para la competencia ITC-2019 (por sus siglas en inglés International Timetabling Competetion), el problema se define como un problema de optimización combinatoria y está clasificado como un problema NP-duro. Debido a que no se puede encontrar la mejor solución con algoritmos determinísticos que presentan prueba de optimalidad en un tiempo polinomial, se trata el problema con técnicas heurísticas. A lo largo de los años, variantes de este problema han sido abordadas por muchos investigadores; debido a esto existen intereses por parte de organizaciones como PATAT (por sus siglas en inglés, Practice and Theory of Automated Timetabling), para generar competencias que permitan mejorar el tratamiento del problema y seguir investigando con nuevas características y apegándolo más a la realidad.

La experimentación se realizó con las instancias proporcionadas por PATAT para la competencia ITC-2019. En esta tesis de maestría se presentan dos heurísticas, una constructiva para generar la solución inicial y una búsqueda local que se utiliza como estrategia para encontrar la factibilidad si la heurística constructiva no lo logra. También se desarrolló un algoritmo para la asignación de estudiantes. Se presentan tres tipos de vecindarios con una propuesta de hibridación de los tres. Los resultados obtenidos nos proporcionan 12 soluciones factibles y se logró quedar en el ranking con la posición nueve, donde participaron 263 usuarios de 57 países diferentes en dicha competencia.

Abstract

In the present research thesis, the Timetabling problem for the ITC-2019 competition is addressed. The problem is defined as a combinatorial optimization problem and is classified as an NP-Hard problem. Since the best solution cannot be found with deterministic algorithms that present proof of optimality in a polynomial time, the problem can be addressed with heuristic techniques. Over the years, variants of this problem have been addressed by many researchers; because of this there is interest from organizations such as PATAT (Practice and Theory of Automated Timetabling), to generate skills to improve the treatment of the problem and continue research with new features and closer to reality.

The experimentation was carried out with the instances provided by PATAT for the ITC-2019 competition. In this master's thesis, two heuristics are presented, a constructive one to generate the initial solution and a local search that is used as a strategy to find the feasibility if the constructive heuristic does not achieve it. An algorithm for student assignment was also developed. Three types of neighborhoods are presented with a proposal to hybridize all three. The results obtained provide us with 12 feasible solutions for this instances and we managed to stay in the ranking with the ninth position, where 263 users from 57 different countries participated in this competition.

Glosario

Instancia	Datos de entrada para el procesamiento de un problema determinado.
Timetabling	Calendarización de recursos
PATAT	Practice and Theory of Automated Timetabling, son conferencias internacionales de Timetabling que tiene lugar cada dos años a nivel mundial.
ITC-2019	International Timetabling Competition, Competencia Internacional de Timetabling que inicio en el año 2019.
University Course Timetabling Problem	(UCTP) Problema de Programación de Cursos Universitarios
Examination Timetabling Problem	(ETP) Problema de Programación de Exámenes
School Timetabling Problem	(STP) Problema de Programación de Horarios Escolares

Contenido

Índice de figuras

Índice de tablas

1. Introducción

En esta tesis se aborda el problema de programación de cursos universitarios jerárquicos aplicando heurísticas de búsqueda local, constructiva y un algoritmo para la asignación de estudiantes. Se trabaja con las instancias del PATAT (por sus siglas en inglés, Practice and Theory of Automated Timetabling), que se encuentra en Internet de la competencia ITC-2019 (International Timetabling Competition 2019).

1.1. Antecedentes

En el campo de la informática existe un área conocida como complejidad algorítmica, que es una medida teórica disponible para comparar diferentes soluciones algorítmicas. Es posible observar cómo se comportan los algoritmos cuando intentan resolver un problema complejo utilizando este campo. La teoría de la complejidad clasifica el universo de problemas de acuerdo con la complejidad inherente de resolverlos. Si existe un algoritmo exacto de tiempo polinomial que permita resolver un problema, entonces el problema es fácil de resolver y se le denomina problema polinomial de tipo P [Cruz, 2016]. Por otra parte, determina que, si un problema no se puede resolver en un tiempo polinomial, se le considera como un problema completamente intratable, a este tipo de problemas se les denomina NP-Completos [Aho, 1974].

Hasta la fecha aún no existen métodos exactos que permitan encontrar la mejor solución a los problemas NP-completos, por lo que un camino viable han sido las heurísticas para "resolver" dichos problemas. Dentro de estos problemas, existe el de horarios escolares, también conocido como: *Timetabling* [Alonso,2008].

Timetabling es uno de los problemas que ha atraído la atención de muchos investigadores a lo largo de los años. Donde se encuentra el problema de programación de cursos universitarios que es un problema de optimización combinatoria. El diseño de horarios de cursos para instituciones académicas siempre ha sido un desafío durante los últimos años [Rezaeipanah, 2020].

Existen complejidades para el problema de programación de horarios universitarios, en primer lugar, el proceso de calendarización es un problema NP-completo, por lo tanto, no se puede resolver en un tiempo polinomial, debido al crecimiento exponencial del problema. En segundo lugar, el número de restricciones duras y suaves en este problema difieren de una institución a otra por lo que es casi imposible plantear un modelo que sirva para todas las universidades [babei, 2015].

Los problemas de calendarización tienen una amplia gama de aplicaciones en educación, deporte, planificación de mano de obra y logística. Una definición más formal de programación de horarios es la siguiente: consiste en asignar una secuencia de eventos (cursos, clases) entre maestros y estudiantes en un periodo de tiempo, satisfaciendo un conjunto de restricciones. Existe una gran variedad en los problemas de programación de horarios, que se basan en el tipo de institución involucrada y el tipo de restricciones [Schaerf,1999].

El problema de horarios universitarios se puede dividir en dos categorías principales, cursos y exámenes, la diferencia más común de estas categorías son:

- Los exámenes deben ser asignados de tal manera que un estudiante no esté presenté en más de un examen al mismo tiempo; en los cursos usualmente deben programarse antes de que se sepa la demanda de los estudiantes.
- A menudo el espacio es una limitación, los exámenes pueden compartir aulas, a diferencia de los cursos que solo son asignados en un aula [Burke, 1997].

La programación de horarios universitarios se ha estudiado tradicionalmente como un problema operativo donde el objetivo es asignar conferencias a salas y franjas horarias, y como principal función crear horarios de alta calidad para estudiantes y profesores [Lindahl, 2018].

El problema de programación de cursos universitarios surge cada año académico, y debe ser resuelto por el personal académico con o sin una herramienta tecnológica que facilite la tarea. Debido a su grado de complejidad, año con año van surgiendo investigaciones y nuevas propuestas de cómo resolver dicho problema. [Thepphakorn¸2020]

En esta tesis nos enfocamos en el diseño de horarios escolares, específicamente en la asignación de cursos universitarios. El objetivo del problema de cursos universitarios es asignar cada una de las clases del curso, a una determinada aula y a un horario, respetando una serie de restricciones duras. Así como asignar a todos los estudiantes a los cursos solicitados, respetando la estructura del curso.

1.2. Planteamiento del problema

El diseño de horarios para instituciones de educación superior presenta una gran problemática debido al elevado número de cursos ofertados, al número reducido de maestros y el bajo número de aulas donde pueden ser asignadas las clases.

Con el surgimiento de diversos algoritmos para automatizar el diseño de horarios universitarios, más instituciones están interesadas en emplear la optimización para generar sus horarios. Debido a que el diseño automático le lleva una gran ventaja al diseño manual, respecto al tiempo y a la calidad de la solución. Lo anterior conlleva a que diversas instituciones y universidades estén interesadas en motivar a los investigadores en seguir trabajando con el problema de horarios escolares [Alonso, 2008].

En esta tesis se trabaja con las instancias diseñadas por la institución PATAT de la competencia del 2019, la cual se enfocó en el problema de diseño general de horarios universitarios. La organización busca hacer los problemas más atractivos, quitando aspectos menos importantes en las instancias reales, manteniendo la complejidad computacional de estos problemas. Al inicio PATAT solo trabajaba con datos de la universidad de Purdue en los Estados Unidos, pero gracias a UniTime, un proyecto de código de abierto se puede tener acceso a muchos problemas de horarios en diversos países. Con esto se llegó al acuerdo para poder trabajar con diversas universidades, y así plantear un problema más complejo, debido a que se formula un problema que contiene características particulares de universidades de diversos países, entre ellas:

- Purdue University, Estados Unidos
- Masaryk University, Republica Checa
- AGH University of Science and Technology, Polonia
- Lahore University of Management Sciences, Pakistán
- İstanbul Kültür University, Turquía
- Bethlehem University, Palestina
- Universidad Yachay Tech, Ecuador
- Turkish-German University, Turquía
- University of Nairobi, Kenya
- Maryville University, Estados Unidos

El problema consiste en un conjunto de aulas, cursos, una serie de restricciones de dispersión de cursos y la demanda de los estudiantes. El objetivo es encontrar un espacio factible para

cada clase, así como la asignación de estudiantes basada en su demanda, respetando una serie de restricciones y preferencias [PATAT, 2018].

Cada instancia tiene un número específico de semanas, cada día tiene ciertos periodos de tiempo. Las aulas cuentan con una capacidad específica y con horarios no disponibles, así como un tiempo de traslado entre cada aula. Los cursos tienen una estructura jerárquica compleja. Los estudiantes cuentan con una demanda individual de los cursos a los que desean asistir. Por último, se tiene una serie de restricciones duras y suaves, donde las duras deben ser satisfechas, para poder encontrar la factibilidad y se debe de minimizar las restricciones suaves violadas para mejorar la calidad de la solución.

1.3. Objetivos

El objetivo principal de esta tesis es proponer y desarrollar un algoritmo para la resolución del problema del ITC 2019 en la parte de factibilidad, utilizando un algoritmo constructivo voraz y un algoritmo de búsqueda local.

A continuación, se detallan los objetivos específicos:

- Determinar los criterios para implementar el algoritmo constructivo voraz (tipo de ordenamiento).
- Implementar el algoritmo constructivo voraz para la resolución del problema del ITC-2019 y validar resultados.
- Implementar el algoritmo de búsqueda local, como estrategia para encontrar factibilidad.
- Implementar un algoritmo para la asignación de estudiantes.

1.4. Alcances y limitaciones

Alcances

- Encontrar la factibilidad en al menos diez instancias de las treinta disponibles.
- Implementar un algoritmo constructivo voraz.
- Implementar una búsqueda local que ayude a mejorar el resultado del algoritmo voraz.

Limitaciones

- No se pretende encontrar factibilidad en todas las instancias debido a su complejidad.
- El algoritmo de asignación de estudiantes sólo buscará la factibilidad, sin tratar de optimizar la violación de restricciones suaves.

1.5. Hipótesis

Aplicando un algoritmo voraz y una búsqueda local al menos se encontrarán 10 soluciones factibles para las instancias de la competencia de ITC-2019.

1.6. Justificación

Una razón por la que existe una gran variedad de métodos para tratar de resolver el problema de Timetabling, es debido a la gran variedad de formulaciones para el problema, dado que las universidades poseen características que las hacen diferentes, y continuamente van cambiando. Por lo tanto, es necesario modificar o adaptar los algoritmos aplicados con anterioridad o incluso diseñarlos desde cero. Esto genera una gran demanda de sistemas para Timetabling, donde cada vez se busca automatizar más el proceso y que sus formulaciones se apeguen más a la realidad, [de Werra, 1985].

Muchos autores creen que el problema de Timetabling no puede ser completamente automatizado, dado que el espacio de búsqueda suele ser enorme, y una intervención humana podría sesgar la búsqueda hacia direcciones prometedoras que el sistema por sí solo no puede encontrar, por lo tanto, la mayoría de los sistemas permiten al usuario al menos ajustar la solución final [Schaerf, 1999]. Por otro lado, comparar los algoritmos diseñados para encontrar soluciones en este tipo de problema es difícil, debido a que no se ha logrado una formulación estándar, por lo tanto, es difícil comparar los resultados obtenidos con otros autores. [Alonso, 2008]. Por lo anterior descrito, sigue siendo importante la investigación y desarrollo de algoritmos para el problema de programación de cursos universitarios.

En esta tesis se abordará el problema de la competencia internacional de horarios 2019 creada por PATAT, que tiene por objetivo, motivar más investigaciones sobre problemas complejos de horarios universitarios que surgen en la práctica. La competencia de PATAT nos ofrece una posibilidad de validar el algoritmo con un conjunto de instancias que contiene distintas características que brindan las universidades y así abrir un nuevo panorama de abordar este tipo de problemas. Dado que el problema aún no se puede solucionar de forma exacta, es necesario diseñar algoritmos que briden soluciones factibles, que se encuentren en un tiempo razonable. El abordar un problema de competencia nos permitirá comparar nuestros resultados con el estado del arte, dado que cuando se aborda un problema específico de una universidad, normalmente sólo se puede comparar con resultados previamente obtenidos por el personal que programa el horario de cursos. También nos proporcionará un verificador online que validará la factibilidad y la calidad de las soluciones obtenidas.

1.7. Organización de la tesis

En el capítulo 2 se presenta el estado del arte del problema, así como una clasificación de las distintas técnicas con las que se ha abordado el problema de Timetabling. En el capítulo 3 se plantea el problema específico de Timetabling que se abordará en la tesis, así como la explicación del modelo matemático. En el capítulo 4 se presenta la metodología, la experimentación y resultados obtenidos. Finalmente, en el capítulo 5 se presenta las conclusiones y el trabajo futuro.

2. Marco teórico

En este capítulo se presenta el estado del arte y una breve descripción del problema de Timetabling, el cual consiste en construir un horario de clases, exámenes o eventos a partir de una serie de recursos tales como profesores, aulas, auditorios, etc. respetando una serie de restricciones de varios tipos. Hace mucho tiempo se sabe que el problema de Timetabling pertenece a la clase de problemas NP-completos, es decir que no se conoce ningún método para resolverlo en un tiempo razonable (polinomial).” [Alonso-Pecina, 2008]. El problema de Timetabling se validó como un problema NP-completo, distintas técnicas han sido aplicadas para este problema. [Cooper, 1995].

El problema está usualmente divido en dos tipos de restricciones:

Duras: Son las restricciones que deben de cumplirse para garantizar un horario factible. Si se quebranta una sola restricción dura, el horario será infactible.

Suaves: las restricciones suaves son menos importante que las duras, y usualmente es imposible no violar todas las restricciones suaves. Se utilizan principalmente para diseñar la función objetivo (Burke, 1997), mientras menos restricciones suaves se violen, más atractiva será la solución.

2.1. Clasificación del problema Timetabling

En la siguiente figura 2.1 se muestra una clasificación del problema de University Course Timetabling. Como se puede observar el problema en general es el de Timetabling, dentro de Timetabling se encuentra el problema de horarios escolares (Educational timetabling), y dentro de los horarios escolares, podemos encontrar al problema de Programación de cursos universitarios (university Course Timetabling Problem UCTP)

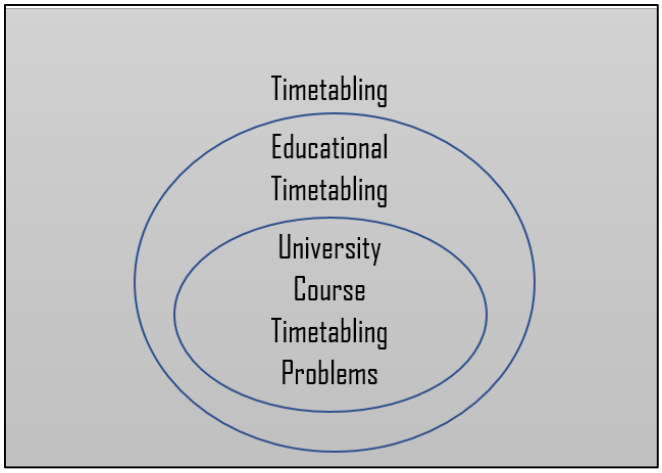

Figura 2.1 Clasificación de Timetabling

Existen tres clasificaciones principales de los problemas de Educational Timetabling:

School Timetabling: es el horario semanal para todas las clases de una escuela primaria, secundaria o preparatoria, evitando que un profesor tenga dos clases en el mismo tiempo, o viceversa.

Course Timetabling: es el horario semanal para todas las clases de un conjunto de cursos universitarios, minimizando el traslape de clases de los cursos con estudiantes en común.

Examination Timetabling: Es la asignación para los exámenes de un conjunto de cursos universitarios, evitando el traslape de exámenes de cursos con estudiantes en común, y extendiendo los exámenes para los estudiantes lo más posible [Schaerf, 1999].

2.2. Definición de University Course Timetabling

El problema de programación de horarios de cursos universitarios consiste en programar un conjunto de sesiones de cada curso con un número de aulas y periodos dados. La principal diferencia con el problema de School Timetabling es que los cursos universitarios pueden tener estudiantes en común, mientras que en los grupos de preparatoria o secundaria son conjuntos disjuntos de estudiantes. [Alonso, 2008]

En algunos casos, el problema de Timetabling consiste en encontrar cualquier horario que satisfaga todas las restricciones, en este caso el problema es formulado como un problema de búsqueda [Schaerf, 1999]. En otras ocasiones el problema es formulado como un problema de optimización, esto significa que es indispensable que se satisfagan todas las restricciones duras y se minimice una función objetivo, que se basa en la violación de las restricciones suaves. Encontrar una solución exacta para el problema, es posible solo en instancias

pequeñas (por ejemplo, menos de 10 cursos), pero en las instancias reales usualmente están envueltos cientos de cursos.

Problema de búsqueda básico

Una formulación del problema de horarios de cursos universitarios:

Hay q cursos $C_l, \dots, C_q$, y cada curso C_i tiene $S_i$sesiones. Hay r retículas $S_1, \dots, S_r$, las cuales son grupos de cursos que tienen estudiantes en común. Esto quiere decir que todos los cursos en $S_l, l = 1, \dots, r$ deben ser programados en horarios diferentes. El número de periodos es $p, y\ l_k$ es el máximo número de clases que pueden ser programadas en el periodo k (es decir, el número de aulas disponibles en el periodo k).

$$\sum_{k=1}^{p} y_{ik} = S_i \qquad (i = 1, \dots, q) \qquad (2.1)$$

$$\sum_{i=1}^{q} y_{ik} \leq l_k \qquad (k = 1, \dots, p) \qquad (2.2)$$

$$\sum_{i \in S_l} y_{ik} \leq 1 \qquad (l = 1, \dots, r; k = 1, \dots, p) \qquad (2.3)$$

$$y_{ik} = 0\ o\ 1 \qquad (i = 1, \dots, m; k = 1, \dots, p) \qquad (2.4)$$

- El conjunto de restricciones en 2.1 establece que cada curso este compuesto del número adecuado de sesiones.
- El conjunto de restricciones en 2.2 hace que en cada periodo no existan más sesiones que aulas.
- El conjunto de restricciones en 2.3 evita que las sesiones en conflicto sean programadas al mismo tiempo [Schaerf, 1999].

Donde en 2.4 las variables toman valores binarios, $y_{ik} = 1$ sí una sesión del curso K_i es programada en el periodo k, en caso contrario $y_{ik} = 0$.

2.3. Métodos para resolver el problema de University Course Timetabling

A lo largo de los años, se han implementado diversos métodos para tratar de resolver este problema, en la figura 2.2, se muestra una breve clasificación de las técnicas indagadas para esta tesis, donde más adelante se hace una breve definición de dichas técnicas y se presentan los trabajos realizados por diversos autores.

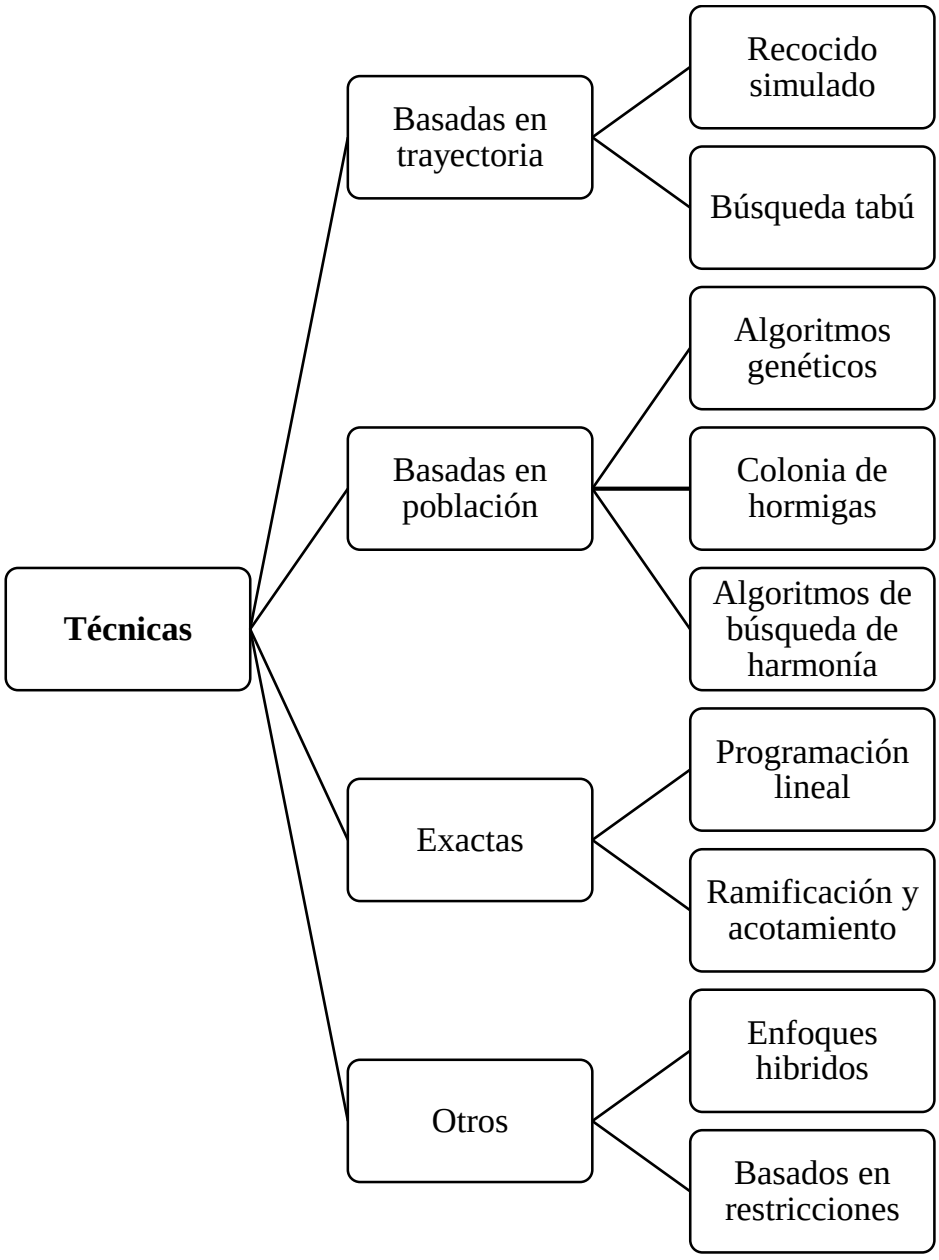

Figura 2.2 Clasificación de técnicas para resolver el problema de UCTP

A continuación, se presenta el estado del arte.

2.3.1. Algoritmos de trayectoria

Este tipo de algoritmos utilizan una única solución para tratar de resolver el problema de optimización. De modo que se genera una solución inicial, que se va manipulando con el fin de obtener una mejora en la solución, hasta llegar a una solución final. El algoritmo termina ocurre cuando se cumple con un criterio de paro.

Búsqueda local

Se han propuesto métodos que van desde los procedimientos de búsqueda local hasta la programación de restricciones. Aunque los procedimientos de búsqueda local son buenos para optimizar la solución inicial factible, también se utilizan como estrategia para encontrar la solución inicial factible **[Francis, 2016]**.

Los algoritmos se mueven de una solución a otra en un espacio de soluciones, mediante el uso de cambios limitados, hasta que se encuentre una solución factible o pase un periodo de tiempo. El algoritmo de búsqueda local comienza a partir de una solución en curso y luego avanza hacia soluciones vecinas, se definen en la adyacencia del espacio de búsqueda del problema y la selección de cada solución para mover, se realiza solo mediante el uso de información sobre soluciones vecinas de manera que maximice el criterio localmente **[Babaei,2015]**.

[Geiger,2010] El artículo presenta un estudio de algoritmos de búsqueda local y un algoritmo de aceptación por umbral, para el problema de horarios. El procedimiento utilizado se basa en una búsqueda local con principios de aceptación por umbral, superando los óptimos locales mediante una aceptación determinista de soluciones inferiores a lo largo de las ejecuciones. Se implementa un vecindario estocástico, eliminado y reasignando eventos al azar.

La presente investigación está enfocada a un caso real del problema de programación de horarios universitarios para la facultad de economía y ciencias de la gestión en Túnez. El objetivo es minimizar el número total de descansos entre clases de cursos. Se desarrollaron un total de 11 vecindarios para tratar de mejorar la calidad del horario. El algoritmo demostró que el enfoque planteado mejoró en un 52% en la eliminación de descansos de la solución a comparación de los horarios planteados por la administración **[Borchani, 2017].**

Recocido simulado

Es un método metaheurístico que ha sido implementado para resolver problemas complejos de optimización. Esto se debe a su facilidad de implementación y capacidad para escapar del optimo local. Inspirado en el calentamiento de metales en la ciencia física. Este enfoque evita el estancamiento en el óptimo local. El método está basado en una búsqueda local con un criterio probabilístico de aceptación, las soluciones obtenidas por la búsqueda reemplazan la solución actual con frecuencia, dependiendo de la mejora de la solución o una función probabilística y esto se repite hasta que se cumplan con los criterios de paro. El proceso del algoritmo es a través de la creación de una solución inicial aleatoria y en puede que, en alguna iteración, la solución actual se reemplaza con una solución aleatoria que puede ser probabilísticamente la solución óptima del problema. Sus componentes principales: temperatura inicial, temperatura final, el parámetro de enfriamiento y una estructura de un vecindario **[Obit, 2016]**.

[Di Gaspero, 2011] En este artículo se presenta una metaheurística para resolver el problema de post-enrolment course Timetabling (PE-CTT), que es uno de los problemas más estudiados de la programación de horarios. La solución inicial se genera con un algoritmo codicioso, ordenando los eventos de tal forma que el más restringido sea el que tenga menos espacios disponibles. Se ocuparon dos vecindarios, uno consiste en mover un evento y el otro en intercambiar dos eventos si es posible.

En el artículo de Basir **[Basir,2013],** reporta el trabajo realizado en la universidad Sains Islam en Malasia como campo de estudio. Se identificaron las restricciones duras y suaves de la universidad, para así poder generar una función objetivo que generara soluciones de calidad utilizando la metaheurística de recocido simulado. La perturbación utilizada es cambiar un evento en espacio-tiempo, y evaluar con la función objetivo. La función objetivo está diseñada con la combinación de las restricciones duras y suaves, de tal forma que a las duras se les da un grado más alto de violación. Fue probado para las instancias del 2002, 2007, donde los resultados obtenidos son inferiores a los de la literatura solo en las instancias 2002.

[Zheng, 2015]En la presente investigación se aborda el problema de cursos universitarios basado en el plan de estudios con restricciones de distancia de viaje entre aulas. Se utilizó el algoritmo de recocido simulado, la motivación de considerar las restricciones de distancia entre eventos consecutivos a los cuales asiste un mismo grupo de estudiantes, la cual es significativa para facilitar el transporte dentro del campus. Se propone un método de dos etapas, una que satisface las restricciones duras en primer lugar y la segunda es minimizar las restricciones suaves. Los resultados obtenidos muestran que el algoritmo de recocido simulado mejorado puede resolver el problema de manera efectiva.

[Gülcü, 2020] Los horarios de los cursos universitarios a menudo se finalizan en etapas, entre las cuales, los cambios en los datos hacen que la versión anterior no sea viable. A medida

que se anuncia cada versión a la comunidad, es deseable tener un horario inicial factible. En esta investigación se implementan dos versiones del problema, en la primera se asume que solo un curso deja de ser factible, y la segunda se asume que varios cursos dejan de ser factibles. El objetivo del algoritmo de recocido simulado multiobjetivo, es encontrar nuevas soluciones frente a los cambios realizados a las soluciones previas. Los extensos experimentos computaciones realizados a los benchmark del ITC-2007 demostraron ser mejores a los algoritmos genéticos implementados previamente.

Búsqueda tabú

Al principio, este algoritmo comienza a moverse desde una solución inicial, luego selecciona el mejor resultado del vecindario, si este resultado no estaba en la lista tabú, el algoritmo se mueve y pondrá el movimiento en tabú, de lo contrario, el algoritmo verificará un criterio de aspiración. Según el criterio de aspiración, si el resultado vecino es mejor que el resultado actual, entonces el algoritmo se moverá hacia ese resultado, incluso si el resultado está en la lista tabú **[Babaei,2015]**.

[Lú, 2008] Se trabaja con un algoritmo de búsqueda tabú adaptativa para resolver el problema de curriculum-based course Timetabling ITC-2007, se propone un algoritmo compuesto por tres fases:

Inicialización: construir una solución factible mediante un algoritmo glotón

Intensificación: se emplea la búsqueda tabú, con dos tipos de vecindario un swap simple y uno más avanzado llamado KempeSwap.

Diversificación: se propone una estrategia aleatoria de perturbación para la solución actual cuando la búsqueda tabú está en un óptimo local. La lista tabú funciona de tal manera, que cuando una sesión se cambia, de una posición a otra, no puede regresar a la primera, y se pone tabú.

2.3.2. Algoritmos de población

En los métodos basados en población, al principio se tienen varias soluciones iniciales, a las que se les denomina población inicial. En cada iteración de los enfoques metaheurísticos basados en la población, se utiliza un mecanismo para seleccionar a las mejores soluciones de la población actual, luego de acuerdo con el método metaheurístico seleccionado, se aplican algunos cambios sobre las soluciones seleccionadas para que se obtenga una mejora en la solución. Este procesamiento continúa hasta llegar a una solución deseable o un criterio de paro que suele ser un número de generaciones (iteraciones).

Algoritmos genéticos

Los algoritmos evolutivos se basan en el modelo informático y se inspiran en el mecanismo evolutivo natural. Actúan sobre una población de posibles soluciones e incluyen tres pasos, selección, regeneración y reemplazo. En la fase de selección, los individuos con una buena aptitud son seleccionadas para ser padres de la próxima generación. En la fase de regeneración se realizan operadores de cruce y mutación en los padres que fueron seleccionados. En la fase de reemplazo, los individuos de la población original son reemplazadas por la nueva población **[Babaei,2015]**.

[Abdullah, 2008] En este artículo se presenta un algoritmo basado en los algoritmos genéticos para resolver el problema de University Course Timetabling. Se utiliza una función de reparación que es totalmente capaz de volver un horario infactible a uno factible, al momento de aplicar un mecanismo de cruce. La solución inicial se genera con una heurística constructiva, para cada individuo de la población. El tamaño del cromosoma es igual al número de eventos asignados, y cada gen representa el periodo para el correspondiente evento. Después de ello, se aplica la selección, el cruzamiento, una mutación y al final la función de reparación.

El problema de programación de cursos universitarios que se aborda en la presente investigación es real, se utilizó un conjunto de datos reales de la facultad de comercio de la Universidad de Alejandría en Egipto. Se implementó un algoritmo genético, el cual utiliza una fórmula de suma ponderada para respetar las preferencias de los profesores y manejar los conflictos. También se emplea una heurística de búsqueda local para obtener una población inicial. Las pruebas demostraron que el algoritmo es una herramienta eficaz para la generación de horarios **[Abdelhalim, 2006].**

Colonia de hormigas

Este método se ha inspirado en el comportamiento de las hormigas para encontrar una ruta entre el lugar del hormiguero y la comida. Las hormigas siempre se mueven aleatoriamente para encontrar comida y luego colocan un rastro de feromona. Otras hormigas cuando encuentran esta ruta la siguen y si alcanzan la comida, regresan a su hogar y colocan un rastro de feromona además de la anterior. La feromona se evapora ligeramente donde los resultados son menos atractivos para la próxima hormiga, por lo tanto, la búsqueda aleatoria se limita a ese criterio. El objetivo de este enfoque es el movimiento de hormigas artificiales para encontrar la ruta más corta.

[Mayer,2012] En este trabajo se presenta una nueva forma de abordar el problema de Post Enrolment Course Timetabling de la competencia 2007. La heurística implementada se basó en el algoritmo de colonia de hormigas (ACO), donde las hormigas construyen soluciones basadas en feromonas e información local. La característica clave de este algoritmo es usar

dos distintas pero simples matrices de feromonas (que representan evento-periodo y evento-aula, respectivamente).

[Azmi,2010] Este articulo implementa un algoritmo de búsqueda de harmonía para abordar el problema de University Course Timetabling. El vector de solución se comporta como la harmonía, cada practica es una iteración, donde se tiene que ir tratando de mejorar la harmonía compuesta. Se puede observar que el algoritmo se comporta de manera eficiente con las instancias grandes.

[Frausto, 2006] En este artículo se implementaron tres metaheurísticas para resolver el problema planteado por PATAT, implementando recocido simulado, búsqueda tabú y algoritmos genéticos, dando mejores resultados recocido simulado para grandes y medianas instancias, y búsqueda tabú para las pequeñas. La solución inicial fue construida con el concepto del evento más restringido. Para la búsqueda tabú se propusieron dos vecindarios, intercambiar dos eventos e intercambiar todos los eventos de dos periodos. Para recocido simulado se seleccionaron aleatoriamente dos periodos y dos aulas, si el intercambio de eventos es factible, el intercambio es actualizado. Si la nueva solución es mejor que la anterior, es aceptada. En los algoritmos genéticos el cromosoma es generado con una heurística determinista.

Varios enfoques metaheurísticos han obtenido buenos resultados cuando se prueban en el conjunto de datos del ITC; sin embargo, pocos han utilizado la técnica de optimización de colonias de hormigas, particularmente en el problema de horarios de los cursos universitarios basados en el plan de estudios del ITC 2007. Este estudio describe un sistema de hormigas que resuelve el problema de horarios de los cursos universitarios basado en el plan de estudios y la calidad del algoritmo se prueba en el conjunto de datos del ITC 2007. El sistema de hormigas pudo encontrar soluciones factibles en todos los casos del conjunto de datos y soluciones cercanas a las óptimas en algunos casos. El sistema de hormigas funciona mejor que algunos enfoques publicados, sin embargo, los resultados obtenidos no son tan buenos como los obtenidos con los mejores enfoques publicados. Este estudio puede utilizarse como punto de referencia para algoritmos basados en hormigas que resuelven el problema de programación de los cursos universitarios basados en el plan de estudios**. [Kenekayoro, 2016]**

2.3.3. Exactos

Programación lineal

Es un método matemático, y se ha aplicado según la estructura de las universidades. El tamaño de la universidad tiene gran influencia en la resolución de este problema, por lo que es difícil obtener las soluciones optimas al final de la realización de este método. Se utiliza principalmente por separado sin combinación de otros enfoques, sin embargo, se pueden

utilizar algunas heurísticas constructivas dentro de este método que facilita el análisis de restricciones.

[Burke, 2008] Este articulo presenta el método de corte y ramifica para una extensión del problema de coloración acotado. Se realiza una formulación de programación entera previamente. Los resultados se implementan en Cplex. En 15 minutos, es posible encontrar soluciones demostrablemente óptimas para dos instancias de la competencia (y buenos limites inferiores para otras instancias ITC 2007).

[Carter, 2001] Describe un sistema integral de programación de horarios y asignación de estudiantes para la universidad de Waterloo entre 1979 y 1985, donde el sistema se basa en la demanda de los estudiantes. El sistema trata de encontrar el mejor horario para maximizar el número de solicitudes satisfechas. El problema se descompone en subproblemas. Cada subproblema se resuelve utilizando una heurística codiciosa y una relajación Langrangiana. El sistema se utilizó con éxito durante 15 años.

[Prabodanie, 2017] Se propone un modelo de programación de enteros binarios para un problema complejo de horarios en una facultad universitaria de Wayamba University que lleva a cabo varios programas de grado. Las variables de decisión se definen con menos dimensiones para economizar el modelo y para mejorar la eficiencia del modelado. Las matrices binarias se utilizan para incorporar las relaciones entre los cursos y los estudiantes, los cursos y los profesores. El modelo incluye restricciones de aplicación general, como integridad, singularidad y consecutividad; y limitaciones específicas del caso. El modelo fue codificado y resuelto con Open Solver, que es un optimizador de código abierto disponible como complemento de Excel. Los resultados indican que los problemas complicados de programación con un gran número de cursos y grupos de estudiantes se pueden formular de manera más eficiente con un menor número de variables y restricciones utilizando el marco de modelado propuesto.

[Phillips,2016]

En el artículo se presenta un enfoque general basado en programación entera para el problema de perturbación mínima en el problema de horarios de cursos universitarios. Se aborda la situación de la universidad de Auckland, la función objetivo maximiza el número total de eventos que se asignan a un periodo de tiempo y un aula. Se demuestra que se puede aplicar en una situación real.

[Pereira, 2016]

Este trabajo presenta un modelo de programación lineal entera que resuelve el problema de programación de cursos universitarios en la facultad de rio de janeiro, Brasil. El modelo fue diseñado para generar soluciones de tal forma que respetara las preferencias de los administradores de la facultad. El modelo supero a las soluciones manuales en términos de

tiempo y calidad. La herramienta fue aprobada por los directores de la facultad y se utiliza desde el segundo semestre del 2011**.**

Basados en restricciones

El método basado en la programación de satisfacción de restricciones es un sistema basado en la informática donde la restricción podría definirse como una limitación en un espacio de soluciones. El objetivo de este método es encontrar un conjunto de valores consistentes donde cada uno de estos valores pueda asignarse a los valores de las variables y satisfacer las restricciones predefinidas **[Babaei,2015]**.

[Muller, 2005] Este articulo implementa un algoritmo de búsqueda directa iterativa (2005, Muller) que está basado en las ideas de los métodos de búsqueda local, sin embargo, en las técnicas clásicas de búsqueda local se trabaja sobre una solución factible, el algoritmo propuesto no es necesario. Trabajar con soluciones incompletas tiene sus ventajas, por ejemplo, cuando el programa no encuentra una solución factible, retorna una gran parte factible. Especialmente en las aplicaciones interactivas, hace ese tipo de asignaciones más fáciles de visualizar.

[Abdullah, 2007] El problema consiste en asignar una lista de eventos a intervalos de tiempo y aulas sujetos a una variedad de restricciones duras y blandas. El método hibrido implementado emplea un algoritmo evolutivo con una búsqueda local. La principal técnica utilizada en el algoritmo evolutivo es una leve mutación siguiente un algoritmo iterativo. El tamaño de la población para el algoritmo genético es de 100, la población inicial se genera con un método parecido al coloreo de grafos aleatorio, añadiendo y removimiento los cursos apropiados para el horario. Se genera la población, después se selecciona el 20% de individuos a mutar, se aplica la mutación y después una búsqueda local, se seleccionan los mejores individuos de la población

[Muller, 2009] El algoritmo de búsqueda consiste en distintas fases: En la fase de construcción, se busca una solución factible utilizando un algoritmo de búsqueda iterativa (IFS), este algoritmo utiliza estadísticas basadas en conflictos para evitar ciclarse. Después se encuentra el óptimo local con una búsqueda local y finalmente para mejorar la solución se utiliza la técnica de recocido simulado. Este artículo proporciona una descripción de un conjunto de soluciones aplicadas satisfactoriamente por el autor de las instancias del problema de International Timetabling Competition 2007.

Un caso real, es el que se aplicó en la universidad autónoma del estado de Morelos, específicamente en la facultad de ciencias químicas. La instancia que se utilizó, tiene 1514 eventos, 41 aulas con 7 características, y un total de 1426 estudiantes. En la investigación se propuso un modelo matemático de satisfacción de restricciones enfocado a una instancia real. Se utilizó un algoritmo constructivo para obtener soluciones del modelo propuesto. Se comparó con los horarios programados por parte de la administración de la universidad,

donde el algoritmo mejoro la asignación del horario para que el uso de las aulas sea más eficiente **[Cruz, 2016].**

El articulo presenta la implementación de la técnica programación de restricciones aplicada en el problema de asignación de cursos universitarios, en el campus Internacional de Malaysia, Malasia. El problema tiene varias restricciones que se deben cumplir, esta investigación no toma en cuenta las restricciones suaves involucradas. Se probó en tres conjuntos de datos del mundo real, tomando en cuenta tres semestres de dicha universidad (2014/2016). El resultado de la investigación muestra que el algoritmo es capaz de generar una solución factible en un corto periodo de tiempo, sin violar ninguna restricción suave, lo cual es aplicable a la universidad **[Junn, 2017].**

Enfoques híbridos

Este grupo de métodos para resolver los problemas de optimización se ha vuelto más atractivos. En estos métodos, se aplican características eficientes de otras técnicas y algoritmos juntos para resolver problemas como el de horarios. Dado que los métodos anteriores tienen algunas debilidades, la combinación de ellos podría eliminar la debilidad. Por ejemplo, un enfoque hibrido, sería que, en la primera fase, la solución inicial se crea utilizando la técnica de programación de restricciones y en la segunda fase el método de recocido simulado para mejorar la solución inicial generada **[Babaei,2015].**

[Alonso, 2008] En este trabajo se presenta un enfoque hibrido entre recocido simulado y búsqueda tabú. Para la solución inicial se utiliza una heurística del evento más restringido, también llamado concepto de saturación. Se considera un evento más restringido que otro, cuando tiene menos periodos disponibles para ser asignado. Después de ello se aplica una hibridación, es decir que primero se ejecuta el algoritmo de recocido simulado, la mejor solución obtenida se pasa como entrada al algoritmo de búsqueda tabú. La vecindad de recocido simulado consiste en seleccionar aleatoriamente dos periodos y dos aulas, que se intercambian, si el intercambio no genera infactiblidad en la solución, se efectúa. Para búsqueda tabú la vecindad consistía en intercambiar dos eventos sin violar ninguna restricción dura y también mover un evento a un lugar vació sin violar ninguna restricción dura. Este algoritmo fue de los mejores para encontrar soluciones factibles en el benchmark UCTP 2002/2007.

[Cambazard,2010] En este artículo se presenta un enfoque de búsqueda local y programación de restricciones para el problema de programación de horarios. Se utilizan cinco diferentes tipos de vecindario, mover un evento a una posición disponible, intercambiar dos eventos, intercambias dos intervalos de tiempo, dos de matching. Los últimos dos consisten, el primero reasigna los eventos en un periodo dado para minimizar el número de eventos asignados a un aula infactible, permitiendo violaciones, el máximo matching es resuelto, los eventos que no son asignados en el matching son asignados en otras aulas

Matching(Ma). TrE + Ma, cambia un evento a un periodo y verifica que no viole las restricciones del aula al checar el problema de matching, si el matching no es factible, el movimiento es rechazado.

En esta investigación se presenta un algoritmo de búsqueda local basado en clases cromáticas para la programación de cursos universitarios. Se discuten varios modelos para resolver el problema. El objetivo principal es que a través de un algoritmo heurístico se encuentren las clases cromáticas para los eventos, donde los vértices del grafo corresponden a los conflictos entre los eventos. Los resultados mostraron que el algoritmo genera soluciones de calidad. **[Kralev, 2017].**

En la investigación se aborda el problema de PECTT, (problema de programación de cursos posterior a la inscripción). Se combinan diferentes algoritmos de búsqueda local en un procedimiento iterativo de dos etapas. En la primera etapa, se usa una búsqueda tabú para generar soluciones factibles. En la segunda etapa, se propone recocido simulado para mejorar la calidad de las soluciones factibles, se trabajó con las instancias propuestas por la competencia internacional de timetabling del 2002 y 2007 **[Goh, 2017].**

[Song, 2018] El objetivo de este artículo es encontrar soluciones factibles para los distintos problemas de asignación de horarios universitarios tomando en cuenta solo las restricciones duras. Se implementó recocido simulado con una búsqueda local. Aquí se realiza un swap entre dos periodos, el cual consiste en intercambiar los eventos de cada periodo, con el propósito de abrir un nuevo espacio para los eventos que no han sido asignados, siempre y cuando se respete la factibilidad. Este algoritmo ha sido de los mejores, encontrando 58 soluciones factibles de 60 del UCTP 2002/2007.

En la presente investigación se implementa un método hibrido basado en el algoritmo genético paralelo y una búsqueda local para resolver dicho problema. El enfoque de búsqueda local se utiliza para tratar de mejor las soluciones encontradas por el algoritmo genético. Garantiza que no se viola ninguna restricción dura, el desempeño del algoritmo se prueba con las instancias del ITC-2007, los resultados obtenidos confirman la eficacia de dicha hibridación y la superioridad del algoritmo en algunas instancias con otros métodos implementados anteriormente **[Rezaeipanah,2020].**

En la presente investigación se desarrolló una herramienta hibrida basada en el algoritmo de búsqueda de cuco para minimizar los costos operativos totales de la universidad. La herramienta se aplicó para resolver once instancias obtenidas de la facultad de ingeniería de la Universidad de Naresuan. El algoritmo utiliza tres estrategias, enfoques adaptativos, estrategias de movimiento(Levy) y técnicas de búsqueda local. Se diseñaron y realizaron experimentos de forma secuencial. El análisis estadístico de los resultados computacionales demostró que los algoritmos propuestos superaron significativamente al proceso convencional de dicha universidad **[Thepphakorn, 2020].**

En la investigación se implementa una búsqueda local iterada (ILS) que es un enfoque bien conocido para la optimización discreta, que combina la perturbación y un marco iterativo. En este estudio, se presenta un algoritmo ILS, reforzado por una hiperheurística que genera heurísticas basadas en un número fijo de operaciones de adición y eliminación. El rendimiento de la hiperheurística propuesta se prueba en dos dominios de problemas diferentes utilizando un punto de referencia del mundo real de instancias de programación de cursos de la segunda competencia internacional de horarios. Los resultados muestran que la combinación de operaciones de agregar y eliminar dentro de un marco ILS produce un enfoque hiperheurístico. **[Soria-Alcaraz, 2016]**

En esta investigación se proponen dos versiones justas del popular problema de horarios de cursos basados en el plan de estudios, el problema MMF-CB-CTT y el problema JFI-CB-CTT, que se basan en la equidad máxima-mínima (MMF) y el índice de equidad de Jain (JFI), respectivamente. Para resolver el problema MMF-CB-CTT, se presenta y evalúa experimentalmente un algoritmo de optimización basado en recocido simulado. Se introducen tres medidas de diferencia de energía diferentes y se evalúa su impacto en el rendimiento general del algoritmo. El algoritmo propuesto mejora la equidad en 20 de las 32 instancias estándar en comparación con los mejores horarios conocidos. La formulación del problema JFI-CB-CTT se centra en el equilibrio entre la equidad y las infracciones agregadas de las restricciones blandas. Los experimentos muestran que la equidad a menudo se puede mejorar a costa de solo un pequeño aumento en el monto total de la penalización [**Mühlenthaler, 2016].**

Este artículo propuso un algoritmo codicioso y de fusión genética (GGFA) para resolver el problema de programación de cursos universitarios de manera eficiente, que puede obtener la solución local óptima mediante el uso del algoritmo codicioso y proporcionar una población inicial de alta calidad para el algoritmo genético. Los resultados de la simulación demuestran que el algoritmo de fusión genética y codicioso propuesto en este documento tiene la capacidad óptima más fuerte y la velocidad de convergencia más rápida que el algoritmo genético estándar, que podría resolver el CTP de manera efectiva y logró buenos resultados **[Wang, 2018**].

Este trabajo propone un algoritmo de búsqueda local híbrido de dos fases para abordar el problema PE-CTT. La primera fase se enfoca en encontrar una solución factible, mientras que la segunda fase intenta minimizar las violaciones de restricciones suaves de la solución factible generada. Para la primera fase, se propone un híbrido de búsqueda tabú con muestreo y perturbación con búsqueda local iterada. Se prueba la metodología propuesta en los casos más difíciles de pruebas comparativas de PE-CTT. El algoritmo híbrido funciona bien y los resultados son superiores en comparación con los métodos recientes para encontrar soluciones viables. Para la segunda fase, se propone un algoritmo llamado Recocido simulado con recalentamiento (SAR) con dos corridas preliminares (SAR-2P). El algoritmo SAR se utiliza para minimizar las infracciones de restricciones suaves al explotar la

información recopilada de las ejecuciones preliminares. El algoritmo es competitivo con los estándares establecidos por los métodos recientes. **[Goh, 2020]**

Este artículo se centra en los horarios de las conferencias universitarias en la Facultad de Ciencias de la Computación y Tecnología de la Información (FCSIT), University Malaysia Sarawak (UNIMAS). En este caso de estudio, la preinscripción al curso no es una práctica. Por lo tanto, no existe una estimación precisa sobre el registro de cursos y hace que el planificador experimentado de la facultad organice el horario según el plan de estudios. Sin embargo, el horario basado en el plan de estudios creará muchos cambios después de que haya comenzado el semestre. Además, los estudiantes están aumentando constantemente de un semestre a otro, aunque el número de recursos del lugar sigue siendo el mismo. Debido a todos estos problemas, el objetivo de este estudio es desarrollar un algoritmo computarizado para minimizar el problema de los conflictos y aumentar la utilización del lugar. Se llevó a cabo un algoritmo de preprocesamiento de datos para predecir el registro del curso. Luego, se propone un método heurístico de dos etapas para resolver el problema de horarios de cursos de la facultad por alumno. El simulador fue probado con datos de tres semestres reales de FCSIT. Todas las soluciones de horarios generadas por el simulador son una solución sin conflictos con un mínimo de cursos no asignados. En términos de utilización del lugar, la solución heurística de dos etapas logra asignar exactamente con la demanda hasta el 98%, pero la solución real puede funcionar mejor con solo el 75% **[Sze, 2017].**

3. Planteamiento del problema

En este capítulo se plantea el problema que se aborda en la presente tesis.

3.1. PATAT

La serie internacional de conferencias sobre la práctica y teoría del Timetabling automático, se celebra cada dos años y sirve como foro para una comunidad internacional de investigadores, profesionales y proveedores sobre todos los aspectos de la generación de horarios asistida por computadora.

Las conferencias están basadas en:

- Programación basada en restricciones
- Timetabling educativo
- Sistemas expertos
- Sistemas difusos
- Búsquedas heurísticas
- Métodos híbridos
- Sistemas basados en conocimiento
- Inteligencia artificial
- Programación matemática
- Timetabling deportivo
- Timetabling en salud
- Timetabling en trasporte
- Herramientas y aplicaciones.

Para esta investigación, se utilizarón las instancias que se generaron para la competencia del ITC 2019 (International Timetabling Competition) lanzada por PATAT.

3.2. Descripción de la competencia 2019

El problema consiste en un conjunto de aulas, cursos con una estructura jerárquica, restricciones de distribución y la demanda de cursos por parte de los estudiantes. El objetivo es asignar todas las clases en un horario y aula, así como asignar los estudiantes a los cursos basado en su demanda, respetando las restricciones y preferencias. Cada clase tiene una lista de horarios disponibles y aulas, esto significa que solo se enumeran las aulas que contienen el aforo suficiente para la clase y que cumplen con todas las demás características (tipo de aula, equipamiento, edificio, etc.).

A continuación, se describen los recursos que se toman en cuenta para resolver el problema.

Periodos

Cada instancia tiene un cierto número de semanas, un número de días en cada semana, y un número de periodos de tiempo por día, como se muestra en la tabla 3.1. El ejemplo especifica que se tienen doce semanas y siete días por semana, de lunes a domingo para generar el horario. En todas las instancias de la competencia un periodo es de cinco minutos, son 288 periodos que cubren un día completo.

Tabla 3-1 Especificaciones de la instancia

Nombre	Número de días	Número de semanas	Periodos por día
"Instancia única"	7	12	288

Aulas

Cada aula tiene un identificador específico y una capacidad. Un aula puede tener horarios no disponibles los cuales se deben respetar a la hora de asignar una clase. Los tiempos de traslado distintos de cero entre aulas, se especifican mediante un cierto valor que expresa el número de intervalos de tiempo que se necesita para ir de un aula a otra. Los tiempos de viaje se consideran simétricos. Como se puede apreciar en la tabla 3.2 cada aula viene con la capacidad de estudiantes que puede albergar.

Tabla 3-2 Ejemplo de capacidad de aulas

Aula	Capacidad
1	50
2	100
3	80

En la tabla 3.3 se muestra una matriz que nos indica la distancia simétrica entre aulas, como se puede apreciar para llegar del aula uno al aula dos, nos tomaría un total de dos periodos de tiempo, lo que equivale a un tiempo de 10 minutos.

Tabla 3-3 Distancia entre aulas

Aula/Aula	**1**	**2**	**3**
1	0	2	1
2	2	0	1
3	1	1	0

Estructura de cursos

Los cursos pueden tener una estructura muy compleja de clases. Cada curso tiene su identificador y está compuesto por una o más configuraciones. Cada configuración de igual manera tiene su identificador y está compuesta por una o más subpartes. A su vez cada subparte tiene su identificador y está compuesta por una o más clases.

Para dar un ejemplo más explícito sobre los cursos, en la figura 3.1 está la estructura de un curso, como se puede observar el curso está constituido por dos configuraciones, seminario-laboratorio y teoría-laboratorio. La primera configuración no tiene una jerarquía, consta de dos subpartes, 1_seminario y 2_laboratorio, donde cada subparte consta de dos clases. Si algún estudiante quisiera ingresar a la configuración seminario-laboratorio, podría escoger tanto seminario 1 o seminario 2, y laboratorio 1 o laboratorio 2. Por otro lado, tenemos la configuración 2 teoría-laboratorio, a diferencia de la anterior en esta se presenta una jerarquía, si el estudiante quisiera asistir a esta configuración, si se asigna a teoría 1 forzosamente tendría que asistir a laboratorio 3, debido a que teoría 1 es la clase padre de laboratorio 3, por otra parte, si quisiera asistir a teoría 2 estaría obligado a estar en el laboratorio 4, para no romper la jerarquía de las clases.

Curso	**Curso de Química**	
Configuración	1.Seminario-Laboratorio	2.Teoría- Laboratorio
Subparte	1_Seminario 2_Laboratorio	3_Teoría 4_Laboratorio
Clases	Seminario 1 Seminario 2 Laboratorio 1 Laboratorio 2	Teoría 1 Teoría 2 Laboratorio 3 Laboratorio 4

Figura 3.1. Estructura de cursos

Clases

Como se dijo en la anterior sección, las subpartes están conformadas por clases. Las clases tienen un conjunto de horarios y aulas a las que pueden ser asignadas.

En la tabla 3-4 se puede apreciar el formato de los horarios que se manejan para las clases.

Tabla 3.4 Horarios de una clase

Horario	Días	Semanas	Inicio	Tamaño	Penalización
1	1010100	1111111111111	90	10	20
2	0101000	1111111111111	108	15	0
3	0001000	0101010101010	96	22	2

Horario 1: es equivalente a lunes- miércoles- viernes de 7-30 a 8-20, todas las semanas

Horario 2: es equivalente a martes y jueves de 9:00 a 10:15 todas las semanas

Horario 3: es equivalente a jueves de 8:00 a 9:50 Semanas pares.

Estudiantes

Cada estudiante cuenta con su plan de estudio, por lo tanto, se trabaja con la demanda individual de los estudiantes. Esto significa que cada estudiante tiene una lista de cursos que necesita atender. Un conflicto de traslape ocurre cuando un estudiante necesita atender dos clases que se dan en el mismo horario.

Un estudiante debe atender una clase por cada subparte de la configuración seleccionada por parte del curso, para satisfacer su demanda. Si existe una relación entre clases de padre-hijo esta se debe de respetar. También cabe destacar que el número de estudiantes que atienden una clase, debe estar limitado por la capacidad de la clase.

Para mayor detalle de esta sección puede consultar el apéndice A, que muestra el formato de lectura en forma de código.

Restricciones de distribución.

Existen varias restricciones de distribución, se listan en la figura 3.2. Cada restricción puede afectar el tiempo del día, los días de la semana, las semanas del semestre o el aula asignada. Cada una de estas restricciones puede ser dura o suave. Las duras no se pueden violar y están marcados con la palabra "*required*". Las suaves pueden ser no satisfechas y hay una penalización para cada violación

Las restricciones están divididas en dos secciones. En la primera sección se evalúan entre pares de clases individuales, por ejemplo, si se deben colocar tres clases al mismo tiempo de inicio, se viola dicha restricción si dos de las tres comienzan en diferentes periodos. Las restricciones de la última sesión de la tabla deben considerar todas las clases, para que puedan ser evaluadas.

Constraint	Opposite	Time	Days	Weeks	Room	Pairs
`SameStart`		√	–	–	–	√
`SameTime`	`DifferentTime`	√	–	–	–	√
`SameDays`	`DifferentDays`	–	√	–	–	√
`SameWeeks`	`DifferentWeeks`	–	–	√	–	√
`SameRoom`	`DifferentRoom`	–	–	–	√	√
`Overlap`	`NotOverlap`	√	√	√	–	√
`SameAttendees`		√	√	√	√	√
`Precedence`		√	√	√	–	√
`WorkDay(S)`		√	√	√	–	√
`MinGap(G)`		√	√	√	–	√
`MaxDays(D)`		–	√	–	–	days over `D`
`MaxDayLoad(S)`		√	√	√	–	slots over `S`
`MaxBreaks(R,S)`		√	√	√	–	breaks over `R`
`MaxBlock(M,S)`		√	√	√	–	blocks over `M`

Figura 3.2 Lista de restricciones con sus parámetros

A continuación, se detalla cada restricción involucrada en este problema.

3.3. Restricciones que se respetan entre pares de clases

Para las restricciones de esta sección, es necesario comparar todas las clases entre sí. Por ejemplo, en la figura 3.3, si tenemos tres clases en una restricción, la clase 1 se tiene que comparar con la 2 y la 3, así mismo la clase 2 debe compararse con la clase 3, para satisfacer la restricción.

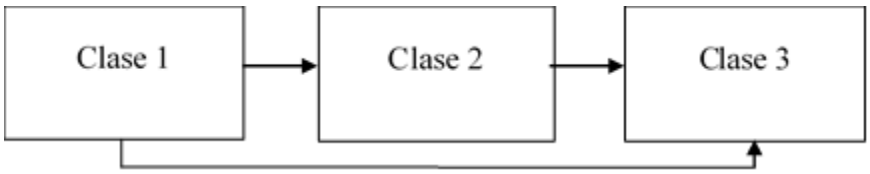

Figura 3.3 Comparación entre clases

A continuación, se detallan las restricciones.

SameStart:

Las clases dadas deben comenzar en el mismo periodo de tiempo. Independiente de sus días o semanas. Esto significa que el inicio de la clase *i* debe de ser igual al inicio de la clase *j*.

Donde $C_i.start$ es el periodo de inicio de la clase i

$$C_i.start = C_j.start$$

SameTime:

Las clases dadas deben ser impartidas en el mismo tiempo del día. Independientemente de sus días o semanas.

$$\left(C_i.start \leq C_j.start \wedge C_j.end \leq C_i.end\right) \vee$$

$$\left(C_j.start \leq C_i.start \wedge C_i.end \leq C_j.end\right)$$

Donde $C_i.lenght$ es el número de periodos que ocupa la clase (tamaño).

Por lo tanto $C_i.end = C_i.start + C_i.length$, que $C_i.end$ representa el final de la clase.

DifferentTime:

Esta restricción es la contraparte de la anterior. Donde nos dice que no deben compartir ningún periodo de tiempo entre clases.

$$(C_i.end \leq C_j.start) \vee (C_j.end \leq C_i.start)$$

SameDays

Las clases dadas deben ser impartidas en los mismos días, sin importar su tiempo de inicio y semanas del semestre.

$$\Big((C_i.days \vee C_j.days) = C_i.days\Big) \vee \Big((C_i.days \vee C_j.days) = C_j.days\Big)$$

Donde $C_i.days$ son los días asignados de la semana de una clase C_i, generando una cadena binaria, donde el uno representa si ese día está asignada, de lo contrario se marca con un cero.

DifferentDays

Las clases deben ser impartidas en diferentes días.

$$(C_i.days \wedge C_j.days) = 0$$

SameWeeks

Las clases deben ser impartidas en las mismas semanas. Donde $C_{i.}$weeks especifica las semanas donde ha sido asignada la clase, es una cadena binaria, donde el uno representa la semana asignada, y el cero lo contrario. Funciona de igual manera que la anterior de SameDays pero ahora con semanas.

$$\Big((C_i.weeks \vee C_j.weeks) = C_i.weeks\Big) \vee \Big((C_i.weeks \vee C_j.weeks) = C_j.weeks\Big)$$

DifferentWeeks

Las clases no deben compartir alguna semana.

$$(C_i.weeks \wedge C_j.weeks) = 0$$

Overlap

Las clases dadas deben traslaparse en tiempo. Dos clases se traslapan en tiempo cuando ellas se coinciden en el tiempo del día, días de la semana, así como en sus semanas.

$$\begin{aligned}\big(C_j.start < C_i.end\big) \wedge \big(C_i.start < C_j.end\big) \wedge \Big(\big(C_i.days \wedge C_j.days\big) \neq 0\Big) \\ \wedge \Big(\big(C_i.weeks \wedge C_j.weeks\big) \neq 0\Big)\end{aligned}$$

NotOverlap

Las clases dadas no deben traslaparse en tiempo.

$$\begin{aligned}\big(C_i.end < C_j.start\big) \vee \big(C_j.end < C_i.start\big) \vee \Big(\big(C_i.days \wedge C_j.days\big) = 0\Big) \\ \vee \Big(\big(C_i.weeks \wedge C_j.weeks\big) = 0\Big)\end{aligned}$$

SameRoom

Las clases dadas deben ser programadas en la misa aula.

$$C_i.room = C_j.room$$

DifferentRoom

Las clases deben ser programadas en distintas aulas.

$$C_{i.}room \neq C_{j.}room$$

SameAttendees

Las clases dadas no deben traslaparse en tiempo, y si son asignadas en un traslape de días de la semana, y semanas, deben ser programadas de tal manera que se le dé el tiempo adecuado para llegar de una clase a otra.

$$\begin{aligned}&\big(C_i.end + C_i.room.travel\big[C_j.room\big] \leq C_j.start\big) \vee \\ &\big(C_j.end + C_j.room.travel[C_i.room] \leq C_i.start\big) \vee \\ &\Big(\big(C_i.days\ and\ C_j.days\big) = 0\Big) \qquad \vee \\ &\Big(\big(C_i.weeks\ and\ C_j.weeks\big) = 0\Big)\end{aligned}$$

Donde $C_i.room.travel\big[C_j.room\big]$ nos indica el valor de periodos de tiempo para llegar del aula *i* al aula *j*.

Precedence

Las clases dadas deben estar una antes que otra, en orden de la información que nos indique la restricción. La primera clase puede iniciar en la semana más temprana, de no ser así, al menos deben iniciar en la misma semana, y la primera clase puede iniciar el día más temprano. En dado caso de que inicien en la misma semana y mismo día, la primera clase debe iniciar en el periodo más temprano que las otras, como se muestra a continuación.

$$\begin{aligned}&\Big(first(C_i.weeks) < first\big(C_j.weeks\big)\Big)\\ &\quad \vee \Big[\big(first(C_i.weeks) = first(C_{j.}weeks)\big)\\ &\quad \wedge \Big[\Big(first(C_i.days) \ < first\big(C_j.days\big)\Big)\\ &\quad \vee \big((first(C_i.days) = first\big(C_j.days\big) \wedge (C_i.end \ < C_j.start)\big)\Big]\Big]\end{aligned}$$

Donde la función *first* nos retorna el primer uno en la cadena binaria.

Para la restricción de precedencia se presentan a continuación varios ejemplos. La precedencia en días se podría cumplir de la siguiente manera, como se muestra en la tabla 3.5. Donde la clase 1, está asignada en días anteriores a la clase 2 y 3, y la clase 2 está asignada un día antes a la clase 3, con esto se estaría cumpliendo la restricción.

Tabla 3-5 Precedencia en días

Clase	Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
1	1	1	0	0	0	0	0
2	0	0	1	0	0	0	0
3	0	0	0	1	0	0	0

En dado caso que las clases compartan el mismo día y las mismas semanas, la restricción se debe cumplir como se muestra en la tabla 3.6, ahora respetando la precedencia en periodos.

Tabla 3-6 Ejemplo de precedencia en periodo de tiempo

Clase	Inicio	Final
4	96	120
5	122	138
6	164	176

Workday(S)

No debe haber más de S intervalos de tiempo entre el inicio de la primera clase y el final de la última clase en un día determinado. Esto se debe validar en las clases que se traslapan en días y semanas.

$$\Big(\big(C_i.days \wedge C_j.days\big) = 0\Big) \vee \Big(\big(C_i.weeks \wedge C_j.weeks\big) = 0\Big) \\ \vee \big(max\big(C_i.end, C_j.end\big) - min\big(C_i.start, C_j.start\big) \leq S\big)$$

MinGap(G)

Cualquiera de las dos clases que se imparten en el mismo día (que comparten días y semanas) deben estar separadas al menos G periodos. Esto significa que debe haber al menos G ranuras entre el final de la clase anterior y el comienzo de la clase posterior, es decir:

$$\Big(\big(C_i.days \wedge C_j.days\big) = 0\Big) \vee \Big(\big(C_i.weeks \wedge C_j.weeks\big) = 0\Big) \\ \vee \big(C_i.end + G \leq C_j.start\big) \vee \big(C_j.end + G \leq C_i.start\big)$$

3.4. Diversas Restricciones

Aquí se presentan el resto de las restricciones, pero a diferencia de las anteriores, estas validan la restricción con todas las clases al mismo tiempo.

MaxDays(D)

Las clases dadas no pueden extenderse a más de "D" días de la semana. Esto significa que el número total de días tiene que ser al menos igual a "D" o menor.

Donde la función *OrFunction* retorna 1 si al menos un bit en la columna es diferente de cero.

$$OrFunction(C_1.days \vee C_2.days \vee \dots C_n.days) \leq D$$

MaxDayLoad(S)

Las clases dadas deben extenderse a lo largo de los días de la semana de manera que no haya más de un número determinado de S franjas horarias cada día. Donde la función *DayLoad* nos retorna el número de periodos asignados a un determinado día (*d*) y a una determinada semana (*w*).

$$DayLoad(d, w) \leq S$$

Donde:

$$DayLoad(d, w) = \sum_i \{C_i.lenght \vee (C_i.days \wedge 2^d) \neq 0 \wedge (C_i.weeks \wedge 2^w) \neq 0\}$$

MaxBreaks (R, S)

Esta restricción limita el número de descansos durante un día, entre el conjunto de clases dadas (no más R descansos durante un día).

Dos clases se consideran consecutivas si están en el mismo bloque, si la brecha entre ellas no es más que S intervalos de tiempo.

$$MergeBlocks(\{(C.start, C.end) | (C.days \wedge 2^d) \neq 0 \wedge (C.weeks \wedge 2^w) \neq 0\}) \vee \leq R + 1$$

La función MergeBlocks funciona recursivamente, dos bloques B_a y B_b que se identifican por sus ranuras de inicio y fin que se traslapan o no son más que S ranuras separadas, hasta que no haya más bloques que puedan fusionarse.

$$(B_a.end + S > B_b.start) \wedge (B_b.end + S > B_a.start) \Rightarrow \big(B.start = min(B_a.start, B_b.start)\big) \wedge \big(B.end = max(B_a.end, B_b.end)\big)$$

MaxBlock (M, S)

Esta restricción limita la duración (tamaño) de un bloque de clases consecutivas durante un día (no más de M ranuras en un bloque). Se considera que dos clases están el mismo bloque si la brecha entre ellas no es mayor que S franjas horarias. Para cada bloque, el número de intervalos de tiempo desde el comienzo de la primera clase hasta el final de la última clase no debe ser mayor que M intervalos de tiempo.

$$max \leq M$$

Para mayor detalle de las restricciones puede consultar el apéndice B, que muestra ejemplos de dichas restricciones en forma de código.

3.5. Ejemplo de solución

En el formato de solución se debe especificar lo siguiente. La clase debe tener su id, seguido por una cadena binaria que nos indica los días a donde fue asignada, su inicio, la cadena binaria de semanas, el aula, y una lista de estudiantes que son los que asisten a esa clase. Así como se muestra en la figura 3.4.

Clase	Días	Inicio	Semanas	Aula	Estudiantes
1	1111000	120	111111111	2	1,2
2	1111100	132	111111111	3	7,12
3	0111000	138	111111111	19	4,5,6,7,8
4	1101000	198	111111111	20	11,14,15,16

Figura 3.4 Solución Final

3.6. Instancias

Las características de las instancias de la competencia ITC-2019 se describen en las tablas 3.7, 3.8 y 3.9, como se aprecia las instancias early (tempranas) fueron publicadas el día 14 de Noviembre del 2018, las middle (intermedias) fueron publicadas el día 18 de Septiembre del 2019 y por último las instancias late (tardías) fueron publicadas el 7 de Noviembre del 2019.

Tabla 3-7 Instancias Early

Instancia	Semanas	Cursos	Clases	Aulas	Estudiantes	Duras	Suaves
agh-fis-spr17	16	340	1,239	80	1,641	820	400
agh-ggis-spr17	16	272	1,852	44	2,116	2202	488
bet-fal17	16	353	983	62	3,018	861	390
mary-spr17	16	544	882	90	3,666	3151	796
muni-fi-spr16	15	228	575	35	1,543	645	95
muni-fsps-spr17	19	226	561	44	865	331	69
muni-pdf-spr16c	13	1,089	2,526	70	2,938	1456	570
pu-llr-spr17	16	687	1,001	75	27,018	416	218
iku-fal17	14	1,206	2,641	214	no	2237	665
tg-fal17	14	36	711	15	no	459	42

Tabla 3-8 Instancias Middle

Instancia	Semanas	Cursos	Clases	Aulas	Estudiantes	Duras	Suaves
agh-ggos-spr17	16	406	1,144	84	2,254	820	400
agh-h-spr17	16	234	460	39	1,988	2202	488
lums-spr18	16	313	487	73	no	861	390
muni-fi-spr17	16	186	516	35	1,469	3151	796
muni-fsps-spr17c	15	116	650	29	395	645	95
muni-pdf-spr16	19	881	1,515	83	3,443	331	69
nbi-spr18	13	404	782	67	2,293	1456	570
pu-d5-spr17	16	212	1,061	84	13,497	416	218
pu-proj-fal19	14	2,839	8,813	768	38,437	2237	665
yach-fal17	14	91	417	28	821	459	42

Tabla 3-9 Instancias Late

Instancia	Semanas	Cursos	Clases	Aulas	Estudiantes	Duras	Suaves
agh-fal17	16	1,363	5,081	327	6,925	820	400
bet-spr18	16	357	1,083	63	2,921	2202	488
iku-spr18	16	1,290	2,782	208	no	861	390
lums-fal17	16	328	502	73	no	3151	796
mary-fal18	15	540	951	93	5,051	645	95
muni-fi-fal17	19	188	535	36	1,685	331	69
muni-fspsx-fal17	13	515	1,623	33	1,152	1456	570
muni-pdfx-fal17	16	1,635	3,717	86	5,651	416	218
pu-d9-fal19	14	1,154	2,798	224	35,213	2237	665
tg-spr18	14	44	676	18	no	459	42

A continuación, se presentan las instancias con las que se trabajó en esta tesis y se lograron obtener soluciones factibles.

Tabla 3-10 Instancias resueltas en la tesis

Instancia	Semanas	Cursos	Clases	Aulas	Estudiantes	Duras	Suaves
mary-spr17	16	544	882	90	3,666	3151	796
muni-fi-spr16	15	228	575	35	1,543	645	95
muni-fsps-spr17	19	226	561	44	865	331	69
pu-llr-spr17	16	687	1,001	75	27,018	416	218
lums-spr18	16	313	487	73	no	861	390
muni-fi-spr17	16	186	516	35	1,469	3151	796
muni-fsps-spr17c	15	116	650	29	395	645	95
muni-pdf-spr16	19	881	1,515	83	3,443	331	69
lums-fal17	16	328	502	73	no	3151	796
mary-fal18	15	540	951	93	5,051	645	95
muni-fi-fal17	19	188	535	36	1,685	331	69
muni-fspsx-fal17	13	515	1,623	33	1,152	1456	570

También cabe destacar que durante la competencia existían 263 usuarios de 57 países diferentes [PATAT, 2019], que intentaron resolver este problema como se aprecia en la figura 3.5.

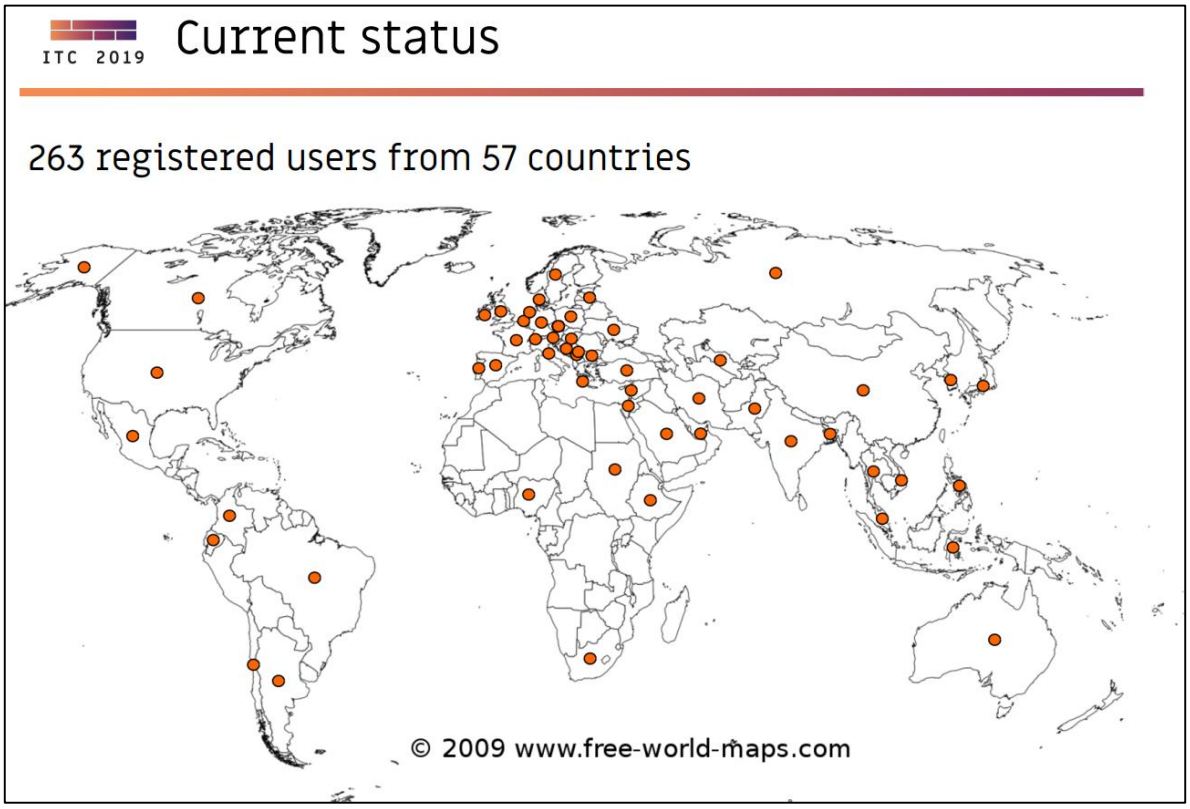

Figura 3.5 Mapa de concursantes de la competencia ITC-2019

3.7. Modelo matemático

En la parte de modelación matemática se tomó el trabajado realizado por "Holm", que sirve para explicar el funcionamiento del problema en general. A continuación, se describe el modelo matemático **[Holm, 2020]**.

3.7.1. Notaciones

A continuación, se describe la notación utilizada para la modelación.

Notación de conjuntos

Símbolo	Descripción
Δ	Conjunto de restricciones de distribución
P	Conjunto de penalización de variables
T	Conjunto de periodos de tiempo
$\mathcal{D}$	Conjunto de días
$\mathcal{W}$	Conjunto de semanas
$\mathcal{K}$	Conjunto de cursos
$\mathcal{C}$	Conjunto de clases
$\mathcal{T}$	Conjunto de horarios
$\mathcal{R}$	Conjunto de aulas
$\mathcal{S}$	Conjunto de estudiantes
$\mathcal{C}_\delta$	Conjunto de clases para una restricción de distribución $\delta \in \Delta$
$\mathcal{C}_s$	Conjunto de clases que un estudiante puede atender $s \in \mathcal{S}$
$\mathcal{C}_\zeta$	Conjunto de clases de una subparte $\zeta \in Z_\omega$
$\mathcal{R}_c$	Conjunto de aulas disponibles para una clase
$\mathcal{T}_c$	Conjunto de horarios disponibles para una clase
$\mathcal{K}_s$	Conjunto de cursos que un estudiante debe atender
$\mathcal{S}_c$	Conjunto de estudiantes que pueden atender una clase
$\mathcal{S}_k$	Conjunto de estudiantes que deben atender un curso
Ω_k	Conjunto de configuraciones de un curso
Z_ω	Conjunto de subpartes de una configuración

Parámetros dados por la instancia.

D	Parámetro de restricción de distribución
G	Parámetro de restricción de distribución
M	Parámetro de restricción de distribución
R	Parámetro de restricción de distribución
S	Parámetro de restricción de distribución
t^{start}	Periodo inicial del horario
t^{lenght}	Duración de periodos del horario
t^{end}	Periodo final del horario
t^{days}	Conjunto de días del horario
$t^{days.first}$	Primer día del horario
t^{weeks}	Conjunto de semanas del horario
$t^{weeks.first}$	Primera semana del horario

Otra notación del modelo

c_i	Una clase especifica con el ID
c_i^{parent}	La clase padre de la clase c_i
c^{limit}	El límite de estudiantes para la clase c
$\tilde{r}$	Un aula “ficticia”, que solo existe en el modelo y no sigue las reglas de un aula normal.
r_i	Una aula de la clase c_i
t_i	Un horario de la clase c_i
$\overline{t}$	Otro horario diferente al $t \in \mathcal{T}, \overline{t} \in \mathcal{T}$
T’	Conjunto de intervalos iniciales
T’’	Conjunto de intervalos finales
M	M-grande

El modelo incluye dos variables de decisión, la de programación del curso y la asignación del estudiante al curso. La variable de programación del curso, es una variable binaria con los índices, de clase, horario y aula. Es definida a continuación:

$$x_{c,t,r} = \begin{cases} 1\ Si\ la\ clase\ c \in \mathcal{C}\ es\ asignada\ al\ horario\ t\ \in \mathcal{T}_c\ en\ el\ aula\ r\ \in \mathcal{R}_c \\ 0\ de\ otra\ manera \end{cases}$$

Las variables de asignación auxiliares para la anterior definición:

$$y_{c,t} = \begin{cases} 1\ si\ la\ clase\ c\ es\ asignada\ al\ horario\ t \\ 0\ de\ otra\ manera \end{cases}$$

$$z_{c,d} = \begin{cases} 1\ si\ la\ clase\ c\ es\ asignada\ al\ día\ d \\ 0\ de\ otra\ manera \end{cases}$$

$$w_{c,r} = \begin{cases} 1\ si\ la\ clase\ c\ es\ asignada\ a\ la\ aula\ r \\ 0\ de\ otra\ manera \end{cases}$$

La variable de asignación del estudiante también es binaria, incluye los índices de estudiante y clase. Es definida de la siguiente forma:

$$e_{s,c} = \begin{cases} 1\ si\ el\ estudiante\ s\ \in \mathcal{S}\ atiende\ la\ clase\ c\ \in\ \mathcal{C}_s \\ 0\ de\ otra\ manera \end{cases}$$

El modelo se muestra a continuación:

1

$$\sum_{t \in \mathcal{T}_c} \sum_{r \in \mathcal{R}_c} x_{c,t,r} = 1 \qquad \forall\, c \in \mathcal{C}$$

2

$$\sum_{\substack{c \in \mathcal{C}, \\ \bar{t} \in \mathcal{T}: \\ t.\texttt{Overlap}(\bar{t})}} x_{c,\bar{t},r} + M \sum_{c \in \mathcal{C}} x_{c,t,r} \leq M \qquad \forall\, r \in \mathcal{R}, t \in \mathcal{T}$$

3

$$\sum_{\substack{t_i \in \mathcal{T}_{c_i}: \\ t_i^{\text{start}} = \tau}} y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ t_j^{\text{start}} \neq \tau}} y_{c_j,t_j} \leq 1 \qquad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, \tau \in \bigcup_{t \in \mathcal{T}_{c_i}} t^{\text{start}}$$

4

$$y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ \neg((t_i^{\text{start}} \leq t_j^{\text{start}} \wedge t_j^{\text{end}} \leq t_i^{\text{end}}) \\ \vee (t_j^{\text{start}} \leq t_i^{\text{start}} \wedge t_i^{\text{end}} \leq t_j^{\text{end}}))}} y_{c_j,t_j} \leq 1 \qquad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

5

$$y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ \neg((t_i^{\text{end}} \leq t_j^{\text{start}}) \\ \vee (t_j^{\text{end}} \leq t_i^{\text{start}}))}} y_{c_j,t_j} \leq 1 \qquad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

6

$$y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ t_i^{\text{days}} \not\subseteq t_j^{\text{days}} \wedge \\ t_j^{\text{days}} \not\subseteq t_i^{\text{days}}}} y_{c_j,t_j} \leq 1 \qquad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

7 $$y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ t_i^{\text{days}} \cap t_i^{\text{days}} \neq \emptyset}} y_{c_j,t_j} \leq 1 \qquad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

8 $$y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ t_i^{\text{weeks}} \not\subseteq t_j^{\text{weeks}} \wedge \\ t_j^{\text{weeks}} \not\subseteq t_i^{\text{weeks}}}} y_{c_j,t_j} \leq 1 \qquad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

9 $$y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ t_i^{\text{weeks}} \cap t_j^{\text{weeks}} \neq \emptyset}} y_{c_j,t_j} \leq 1 \qquad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

10 $$y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ \neg((t_j^{\text{start}} < t_i^{\text{end}}) \wedge \\ (t_i^{\text{start}} < t_j^{\text{end}}) \wedge \\ (t_i^{\text{days}} \cap t_j^{\text{days}} \neq \emptyset) \wedge \\ (t_i^{\text{weeks}} \cap t_j^{\text{weeks}} \neq \emptyset))}} y_{c_j,t_j} \leq 1 \qquad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

11 $$y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ (t_j^{\text{start}} < t_i^{\text{end}}) \wedge \\ (t_i^{\text{start}} < t_j^{\text{end}}) \wedge \\ (t_i^{\text{days}} \cap t_i^{\text{days}} \neq \emptyset) \wedge \\ (t_i^{\text{weeks}} \cap t_i^{\text{weeks}} \neq \emptyset)}} y_{c_j,t_j} \leq 1 \qquad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

12 $$w_{c_i,r_i} + \sum_{r_j \in \mathcal{R}_{c_j} \setminus \{r_i\}} w_{c_j,r_j} \leq 1 \qquad \forall c_i, c_j \in \mathcal{C}_\delta : i < j, r_i \in \mathcal{R}_{c_i}$$

13 $$w_{c_i,r} + w_{c_j,r} \leq 1 \qquad \forall c_i, c_j \in \mathcal{C}_\delta : i < j, c_j \in \mathcal{C}_\delta, r \in \mathcal{R}_{c_i} \cup \mathcal{R}_{c_j}$$

14 $$x_{c_i,t_i,r_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ t_i.\texttt{Overlap}(t_j)}} y_{c_j,t_j} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ \neg t_i.\texttt{Overlap}(t_j), \\ r_j \in \mathcal{R}_{c_j}: \\ t_i.\texttt{Overlap}(t_j,r_j,r)}} x_{c_j,t_j,r_j} \leq 1 \; \forall c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in t_{c_i}, r_i \in \mathcal{R}_{c_i}$$

15 $$y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ t_j^{\text{weeks.first}} < t_i^{\text{weeks.first}} \vee \\ (t_j^{\text{weeks.first}} = t_i^{\text{weeks.first}} \wedge \\ (t_j^{\text{days.first}} < t_i^{\text{days.first}} \vee \\ (t_j^{\text{days.first}} = t_i^{\text{days.first}} \wedge t_j^{\text{start}} < t_i^{\text{start}})))}} y_{c_j,t_j} \leq 1 \qquad \forall c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

16 $$y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ t_i^{\text{weeks}} \cap t_i^{\text{weeks}} \neq \emptyset \wedge \\ t_i^{\text{days}} \cap t_i^{\text{days}} \neq \emptyset \wedge \\ \max(t_i^{\text{end}}, t_j^{\text{end}}) - \min(t_i^{\text{start}}, t_j^{\text{start}}) > \texttt{S}}} y_{c_j,t_j} \leq 1 \qquad \forall c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

17 $$y_{c_i,t_i} + \sum_{\substack{t_j \in \mathcal{T}_{c_j}: \\ \neg(t_i^{\text{weeks}} \cap t_i^{\text{weeks}} = \emptyset \vee \\ t_i^{\text{days}} \cap t_i^{\text{days}} = \emptyset \vee \\ t_i^{\text{end}} + \texttt{G} \leq t_j^{\text{start}} \vee \\ t_j^{\text{end}} + \texttt{G} \leq t_i^{\text{start}})}} y_{c_j,t_j} \leq 1 \qquad \forall\, c_i, c_j \in \mathcal{C}_\delta : i < j, t_i \in \mathcal{T}_{c_i}$$

18 $$\gamma_d = \begin{cases} 1 & \text{Si cualquier clase } c \in \mathcal{C}_\delta \text{ es asignada en un día } d \in \mathcal{D} \\ 0 & \text{De otra manera} \end{cases}$$

19 $$\sum_{c \in \mathcal{C}_\delta} z_{c,d} \leq M \gamma_d \qquad \forall d \in \mathcal{D}$$

20 $$\sum_{d \in \mathcal{D}} \gamma_d \leq \mathtt{D}$$

21 $$\phi_{w,d} = \sum_{\substack{c \in \mathcal{C}_\delta, \\ t \in \mathcal{T}_c : t^{week} = w \wedge t^{day} = d}} t^{\text{length}} y_{c,t} \qquad \forall w \in \mathcal{W}, d \in \mathcal{D}$$

22 $$\phi_{w,d} \leq \mathtt{S} \qquad \forall w \in \mathcal{W}, d \in \mathcal{D}$$

23 $$\beta_{w,d} - 1 \leq \mathtt{R} \qquad \forall w \in \mathcal{W}, d \in \mathcal{D}$$

24 $\rho_{w,d,\tau} = \begin{cases} 1 & \text{Si un bloque no es más grande que M en la semana } w \in \mathcal{W} \text{ en el } d \in \mathcal{D} \text{ día en el periodo } \tau \\ 0 & \text{De otra manera} \end{cases}$

25 $$\sum_{\substack{w \in \mathcal{W}, \\ d \in \mathcal{D}, \\ \tau \in \mathrm{T}}} \rho_{w,d,\tau} = 0$$

26 $$\sum_{s \in \mathcal{S}_c} e_{s,\mathrm{c}} \leq c^{\text{limit}} \qquad \forall\, c \in \mathcal{C}$$

27 $$e_{s,c_i} \leq e_{s,c_j} \qquad \forall\, s \in \mathcal{S}, c \in \mathcal{C}_s : c_i^{\text{parent}} = c_j$$

28 $$\sum_{c \in \mathcal{C}_\zeta} e_{s,c} = 1 \qquad \forall\, k \in \mathcal{K}_s : |\Omega_k| = 1, \omega \in \Omega_k, \zeta \in \mathrm{Z}_\omega, s \in \mathcal{S}_k$$

29 $b_{s,\omega} = \begin{cases} 1 \text{ Si un estudiante } s \in \mathcal{S} \text{ esta asignado a una clase de una configuración } \omega \in \Omega_k \\ 0 \quad \text{De otra manera} \end{cases}$

30 $$\sum_{\omega \in \Omega_k} b_{s,\omega} = 1 \qquad \forall\, s \in \mathcal{S}, k \in \mathcal{K}_s : |\Omega_k| > 1$$

31 $$\sum_{c \in \mathcal{C}_\zeta} e_{s,c} = b_{s,\omega} \qquad \forall\, k \in \mathcal{K}_s : |\Omega_k| > 1, \omega \in \Omega_k, \zeta \in \mathrm{Z}_\omega, s \in \mathcal{S}_k$$

La restricción (1) define que para todas las clases $c \in C$ deben estar programadas en un horario, y que ninguna aula pueda reservarse dos veces o más al mismo tiempo. La restricción (2) asegura que si una clase está programada en el aula r y en el horario t, entonces no puede haber clases programadas horarios que se traslapen con la misma aula. La M grande es el número total de clases M=|C|.

La restricción SameStart (3) dice que si la clase c_i es asignada al horario t_i, la cual empieza en el periodo de tiempo τ, entonces la clase c_j no puede ser programada en un horario con un inicio diferente a ese periodo de tiempo.

La restricción SameTime (4) dice que si una clase c_i es asignada a un horario t_i, entonces la clase c_j no puede ser asignada en un horario que no tenga el mismo horario de c_i.

La restricción DifferentTime (5) dice que si una clase c_i es asignada a un horario t_i, entonces la clase c_j no puede ser asignada en un horario que se traslape con c_i.

La restricción SameDays (6) dice que si una clase c_i es asignada a un horario t_i, con un conjunto de días t_i^{days}, entonces la clase c_j no puede ser asignada a un horario con un conjunto de días t_j^{days}, si el conjunto más pequeño no es un subconjunto del mayor.

La restricción DifferentDays (7) dice que si una clase c_i es asignada a un horario t_i, con un conjunto de días t_i^{days}, entonces la clase c_j no puede ser asignada a un horario con un conjunto de días t_j^{days}, si los conjuntos tienen días en común.

La restricción SameWeeks (8) dice que si una clase c_i es asignada a un horario t_i, con un conjunto de semanas t_i^{weeks}, entonces la clase c_j no puede ser asignada a un horario con un conjunto de semanas t_j^{weeks}, si el conjunto más pequeño no es un subconjunto del mayor.

La restricción DifferentWeeks (9) dice que si una clase c_i es asignada a un horario t_i, con un conjunto de semanas t_i^{weeks}, entonces la clase c_j no puede ser asignada a un horario con un conjunto de semanas t_j^{weeks}, si los conjuntos tienen semanas en común.

La restricción Overlap (10) define que si una clase c_i es asignada a un horario t_i, la clase c_j no puede ser asignada en un horario que no se traslape.

La restricción NotOverlap (10) define que si una clase c_i es asignada a un horario t_i, la clase c_j no puede ser asignada en un horario t_j que se traslape con t_i.

La restricción SameRoom (12) define que si una clase c_i es asignada a un aula r_i, entonces la clase c_j no debe ser asignada en un aula diferente a r_i.

La restricción DifferentRoom (13) define que si una clase c_i es asignada a un aula r_i, entonces la clase c_j no debe ser asignada al aula r_i.

La restricción SameAttendees (14) establece que si una clase c_i, es asignada a un horario t_i en un aula r_i, la otra clase no puede ser asignada de tal forma que se traslape con la anterior (como la restricción de Overlap (10)) pero también que ambas clases respeten el tiempo de traslado de un aula a otra. Eso significa que las clases deben programarse de tal manera que el tiempo de viaje entre ambas aulas asignadas no exceda la duración entre los horarios asignados.

La restricción de Precedence (15) establece que si una clase c_i es asignada a un horario t_i, entonces la otra clase c_j no debe ser asignada a un horario que empiece en una semana más temprana o en un día más temprano de la semana (si empiezan en la misma semana), o en un periodo de tiempo más temprano (si empiezan en la misma semana y el mismo día).

La restricción de Workday(S) (16) define que si una clase c_i es asignada a un horario t_i, entonces la otra clase c_j no debe ser asignada a un horario se traslape en alguna semana o día, de tal forma que la diferencia entre periodos de tiempo del inicio de la clase más temprana y el final de la clase más tarde, sea mayor al parámetro S de periodos que establece la restricción.

La restricción de MinGap(G) (17) define que si una clase c_i es asignada a un horario t_i, entonces la otra clase c_j no debe ser asignada a un horario se traslape en alguna semana o día, de tal forma que el número de periodos entre la hora de finalización más temprana y la última hora de inicio sea menor al número de periodos G.

La restricción MaxDays(D) dice que las clases de la restricción no se deben distribuir en más de D días. Se define una variable auxiliar (18), para todos los días la variable γ_d es limitada por la restricción (19), donde $\mathcal{M} = |\mathcal{C}_\delta|$ representa el número de clases involucradas en la

restricción, la restricción queda entonces de la forma (20) para establecer el límite de días que pueden ser distribuidas las clases.

La restricción MaxDayLoad(S) dice que un conjunto de clases dadas no puede ser asignada de tal manera que el número de periodos de cualquier día no exceda el parámetro S. Se define la carga del día como $\phi_{w,d} \in \mathbb{Z}^+$ en el día d y en una semana w (21). La restricción queda de la siguiente manera (22) donde se verifica que en todos los días y semanas no se exceda con la carga (S).

La restricción (23) MaxBreaks(R, S) define que no puede haber más de R descansos durante un día, en cualquier día y semana. Un descanso se define que tiene más de S periodos de tiempo vacíos entre dos clases consecutivas. La variable $\beta_{w,d}$ nos dice el número de descansos del día d y la semana w, por lo tanto debe ser menor al valor de R descansos.

La restricción MaxBlock(M, S) establece que si un bloque en cualquier día y en cualquier semana no debe ser mayor a M periodos de tiempo. Dos clases consecutivas se consideran en el mismo bloque si no existen más de S periodos de tiempo entre ellas. La variable auxiliar (24) ayuda a saber si existe un bloque más grande que M en un día y semana especifico. Por lo tanto, la restricción queda así (25).

El número de estudiantes asignados a una clase (26) no puede exceder el límite de estudiantes permitido. Si un estudiante está asignado a una clase con una clase padre, la clase padre también debe ser asignada al estudiante (27).

Para los cursos que tienen una sola configuración, los estudiantes que atiende a ese curso deben ser asignadas exactamente a una clase de cada subparte de la configuración (28).

Para los cursos que tienen más de una configuración, se define una variable auxiliar (29), los estudiantes deben atender los cursos en una sola configuración (30). Para asistir a una configuración, los estudiantes deben asistir exactamente a una clase de cada subparte de la configuración. Además, si un estudiante no asiste a una configuración (31), ninguna clase de esa configuración puede estar asignada al estudiante.

4. Metodología

En este capítulo, se presenta la manera de cómo se abordó el problema del ITC2019, se utilizaron tres algoritmos, dos para la asignación de clases y uno para la asignación de estudiantes. Como se aprecia en la figura 4.1, se inicia con la lectura de la instancia, después se aplica un algoritmo constructivo para generar una solución inicial, a la solución inicial se le aplica un algoritmo de búsqueda local para tratar de mejorar la calidad de la solución y así encontrar la factibilidad, por último, se asignan los estudiantes respetando la demanda de los cursos, al final se imprime la solución.

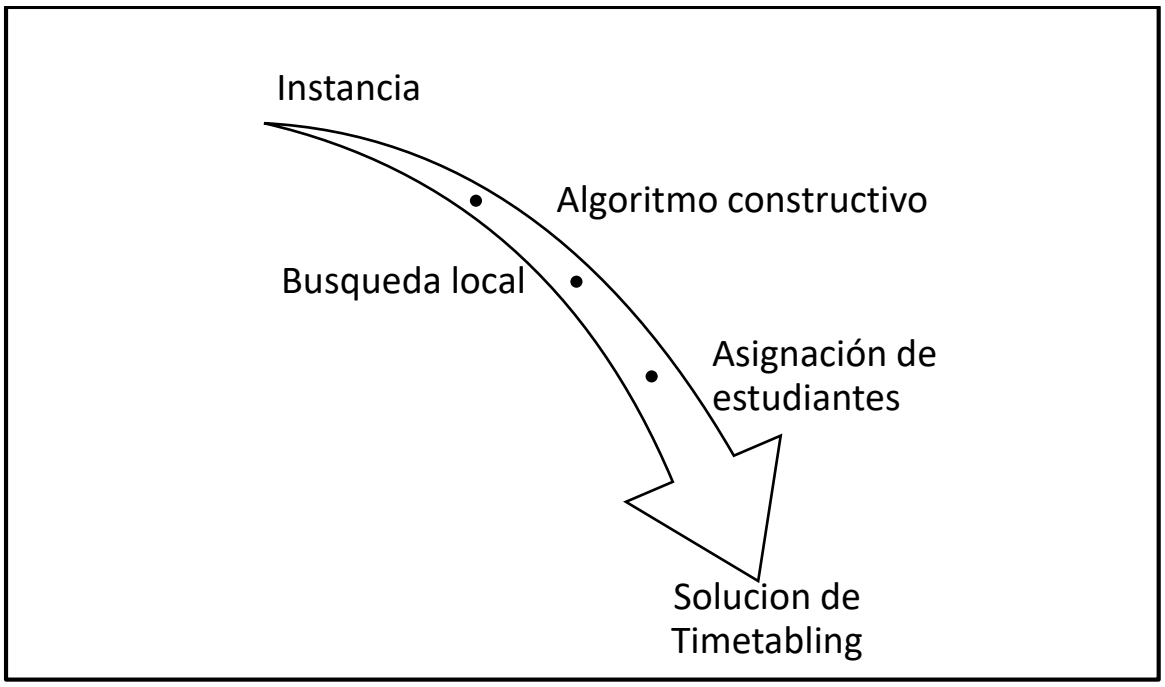

Figura 4.1. Propuesta de solución

4.1. Diseño de la representación de la solución

A continuación, se presenta la función objetivo del problema, así como la representación de la solución y el proceso para calcular las posibilidades que tiene una clase para ser asignada.

4.1.1. Función objetivo

Cabe destacar que esta función objetivo se utilizó como estrategia para reducir el número de clases no asignadas, en el algoritmo de búsqueda local, cumpliendo con todas las restricciones en el modelo matemático que se presentó en la sección anterior.

La función objetivo del algoritmo es:

$$Minimizar\ z = csa$$

$$sujeto\ a\ Hcv = 0$$

$$donde$$

$$csa: es\ el\ número\ de\ clases\ no\ asignadas$$

$$Hcv: es\ el\ número\ de\ restricciones\ duras\ violadas$$

4.1.2. Representación de la solución

Para dar un ejemplo de la representación de la solución tomaremos la clase 1, que tiene tres aulas disponibles y cuatro horarios, por lo tanto, tiene AH combinaciones posibles aula-horario en este caso AH = 12 (3 x 4 = 12). Utilizaremos el número de 0 para indicar que se asigna la clase a la combinación Aula 0, Horario 0, un 1 para indicar que se asigna al aula 0, horario 1 y así sucesivamente. De tal manera que para obtener el horario sacamos el residuo de la división de AH entre el número de horarios y para obtener el aula es el resultado de la división de AH entre el número de horarios:

Aula = AH/(número de horarios)

Horario =AH MOD (número de horarios)

Podemos ver un ejemplo en la tabla 4.1. En caso de que la clase no requiera aula, AH será igual al número de horarios.

Tabla 4.1 Representación de diversas asignaciones para la clase 1 con 3 aulas y 4 horarios

Asignación	Aula	Horario	Asignación	Aula	Horario
0	0	0	6	1	2
1	0	1	7	1	3
2	0	2	8	2	0
3	0	3	9	2	1
4	1	0	10	2	2
5	1	1	11	2	3

Siguiendo el ejemplo anterior, donde la clase tiene doce combinaciones para ser asignada, se aprecia en la figura 4.2, al inicio la clase tiene doce lugares disponibles, después de ser asignada, se calculan los movimientos factibles que tiene aún, en este ejemplo, después de realizar la asignación, la clase aún tiene cuatro lugares donde se puede mover, y cambiar su asignación. Los movimientos siempre son factibles, entonces respetan la disponibilidad de aula y las restricciones duras.

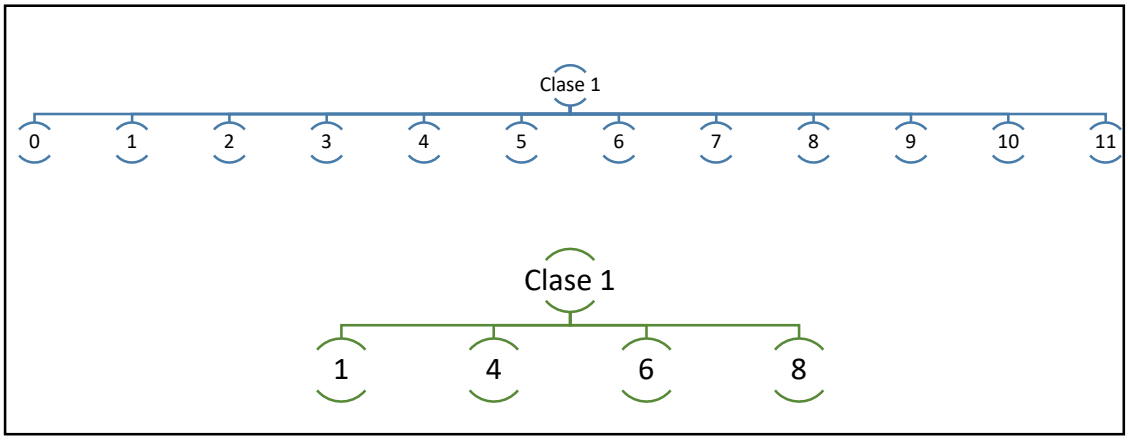

Figura 4.2 Movimientos factibles para cada clase

Para evitar el traslape de clases en las aulas, se utiliza una matriz tridimensional para cada aula. Donde las coordenadas de la matriz nos indican la semana, el día y el periodo de tiempo del aula. Como se muestra en la siguiente figura 4.3, cada posición de la matriz nos indica, si el periodo está ocupado, disponible o no disponible, y así evitar asignar una clase a un periodo ocupado o no disponible.

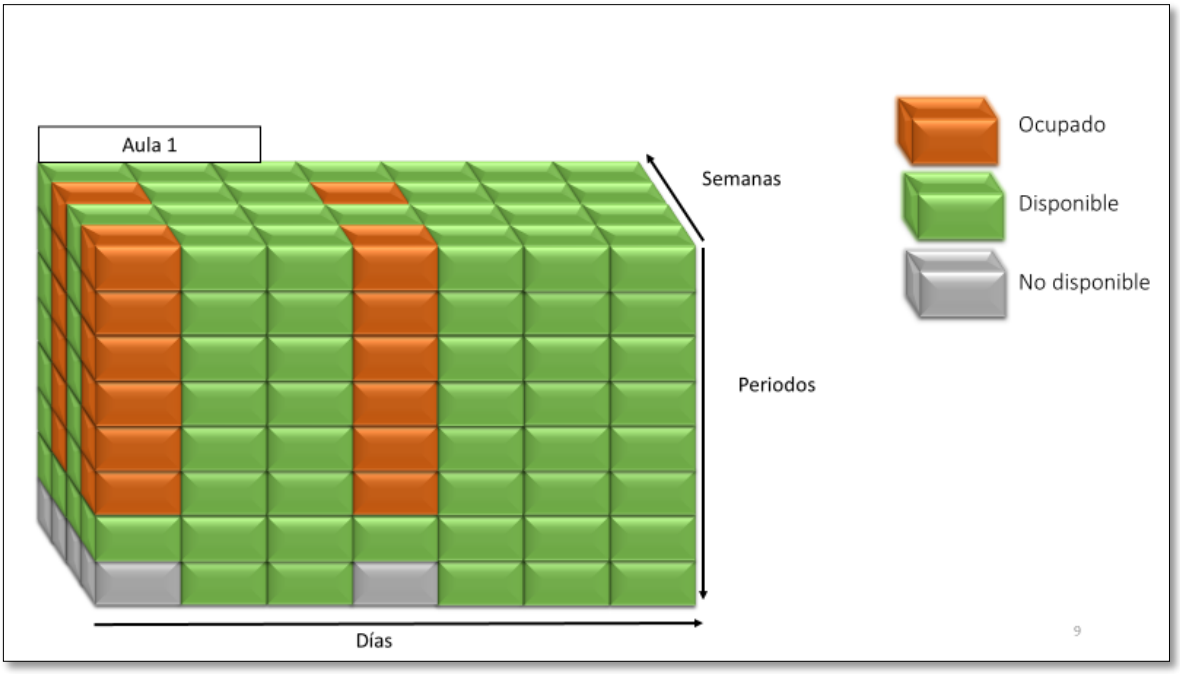

Figura 4.3 Representación de la matriz tridimensional para aulas

Para realizar el anterior proceso, es fundamental la siguiente función, a continuación, se hace un desglose de esta. Se presentan las funciones importantes para poder encontrar la factibilidad en los horarios de las clases.

```
1  Inicio
2  Seleccionar c de C
3  T = contar_horarios()
4  A = contar_aulas()
5  HF ={ }
6  Contador_horarios_factibles=0, contador_horarios=0, contador_aulas=0
7  Mientras contador_horarios<T
8          Mientras contador_aulas<A
9                  Si checar_disponibilidad(contador_horario,contador_aula) Entonces
10                         Si checar_restricciones_duras(clase, virtual_asi) Entonces
11                                 Contador_horarios_factibles++
12                                 HF = HF ∪ virtual_asi
13                         Fin_Si
14                 Fin_si
15                 Contador_aulas++
16         Fin_Mientras
17         Contador_horarios++
18 Fin_Mientras
19 Imprimir Contador_horarios_facibles
20 Fin
```

Figura 4.4. Función checar factibilidad

En la línea (2) de la figura 4.4 se selecciona la clase a la cual se validarán sus horarios factibles, en la línea (3) y (4) se obtiene el número total de horarios y aulas que tiene la clase. En la línea (6) se inicializan los contadores con cero, se recorren las combinaciones con los ciclos de la línea (7) y (8). En la línea (9) se llama a la función checar_disponibilidad, si retorna un valor de uno entonces esa combinación es factible respecto al aula y horario, después de ello se verifica que esa combinación cumpla con las restricciones duras involucradas con la clase (línea 11), si se cumple con ambas, entonces se contabiliza un horario factible y se agrega la combinación en horarios factibles (HF). Al final de la función se obtiene el número total de horarios disponibles para la clase, (línea 19).

```
1   Checar_disponibilidad(contador_horario, contador_aula)
2   R=obtener_aula(contador_aula)
3   D=obtener_dias(contador_horario)
4   W=obtener_semanas(contador_horario)
5   S=obtener_inicio(contador_horario)
6   L=obtener_tamaño(contador_horario)
7   Disponible=1
8   Para d ∈ D
9       Para w ∈ W
10          Para s=S hasta S+L
11              Si aula[R][w][d][s] <>0 entonces
12                  Disponible=0
13                  Retornar Disponible
14              Fin_si
15          Fin_para
16      Fin_para
17  Fin_para
    Retornar Disponible
```

Figura 4.5 Función checar aula

Para checar la disponibilidad de un horario en el aula, se diseña la siguiente función de la figura 4.5. En las primeras líneas se obtiene el aula donde se va a asignar (R), los días del horario (D), las semanas (W), el inicio de la clase (S), y el tamaño de la clase (T). En la línea (7) se inicializa la variable disponible, esta variable es importante porque nos indica cuando el horario está disponible se queda con un valor de uno, de lo contrario tomara un valor de cero. Para checar la disponibilidad en el aula, se recorre una matriz "aula", que nos indica cuando el espacio está disponible, si la matriz tiene un valor de cero indica que el espacio está vacío, de lo contrario, ya se encuentra ocupado por alguna otra clase o ese espacio no está disponible, si es así entonces el valor de disponible cambia a cero y retorna un valor de cero, indicando que ese horario-aula no está disponible.

```
1   Checar_restricciones_duras(clase, virtual_asi)
2   R = leer_restricciones(clase)
3   Asignar(clase, virtual_asi)
4   Factible=1
5   Para r ∈ R
6           CR = obtener_clases_involucradas(r)
7           t=Obtener_tipo_restriccion(r)
8           Si tipo<15 entonces
9                   Para cr ∈ CR
10                          Si Validar_restriccion(clase, cr, t) <>1 entonces
11                                  Factible=0
12                                  retornar Factible
13                          Fin_si
14                  Fin_para
15          Sino
16                  Si Validar_restriccion(clase, CR, t) <>1 entonces
17                          Factible=0
18                          retornar Factible
19                  Fin_si
20          Fin_si
21  Fin_para
21  Eliminar(clase, virtual_asi)
22  Retornar Factible
```

Figura 4.6 Función checar restricciones duras

Después de haber validado la disponibilidad del horario-aula de la clase, es necesario verificar que esa combinación cumpla con las restricciones duras, por lo tanto, se diseñó la función que viene en la figura 4.6. Primero se obtienen las restricciones duras de la clase donde está involucrada, en la línea (3), se asigna la clase para poder comparar con las demás clases. Se inicializa una variable llamada factible, para indicar que la combinación es válida se utiliza el número uno, de lo contrario se asigna un cero, para saber que no es factible. Después de haber extraído las restricciones se recorre una por una (línea 5), para cada restricción se tiene que acceder a las clases que están involucradas y que ya han sido asignadas, después se obtiene el tipo de restricción (línea 7), y se valida si es especial o no, como se han descrito anteriormente, hay varios tipos de restricciones. Para validar que la restricción es por pares o es especial (línea 8), se enumeran las primeras catorce restricciones como pares y las otras como especiales. Si la restricción es por pares, se tiene que ir validando cada clase involucrada de la restricción (línea 9 a línea 14). Si es especial se valida en conjunto (línea 16 a línea 19). Por último, se elimina la asignación virtual, y se retorna un valor de verdadero si la asignación es factible.

.

4.2. Solución inicial

Para generar la solución inicial, se implementa un ordenamiento de clases. Se elige el evento más restringido y se actualizan las posibilidades cada vez que se hace una asignación. Como se muestra en el siguiente diagrama, al momento de seleccionar la clase más restringida, en dado caso de existir un empate, se selecciona aleatoriamente el candidato, después si la asignación es válida, se elimina la clase. Para seleccionar la clase más restringida, se contabilizan el número de espacios disponibles, así la clase que tenga menos espacios disponibles es la clase más restringida. La variable F, almacena el número de clases que no han sido asignadas, que es la función objetivo que planteamos para este problema, el siguiente algoritmo de búsqueda local, tiene el objetivo de disminuir lo máximo posible esa variable, hasta llegar a cero para encontrar una solución factible.

```
1   Algoritmo SolucionInicial
2       F=0
3       contador=0
4       Sol = {Ø}
5       C = todaslasclases
6       contador=asignarClasesFijas(C)
7       Mientras contador< |C| Hacer
8           ordenarClases()
9           Seleccionar la clase c más restringida
10          Si existe lugar disponible para asignar la clase c Entonces
11                  Asignar la clase c
12                  Sol = Sol ∪ c
13          SiNo
14                  ++F;
15                  Eliminar clase;
16          FinSi
17          contador=contador+1;
18      Fin Mientras
19  FinAlgoritmo
```

Figura 4.7 Algoritmo constructivo de la solución inicial

4.3. Búsqueda local

Para buscar una mejora en la solución obtenida por el algoritmo constructivo descrito en la anterior sección, se aplica una búsqueda local para tratar de minimizar el número de clases no asignadas, y así mejorar la solución inicial.

4.3.1. Vecindarios

Para la implementación del algoritmo de búsqueda local, se utilizaron tres vecindarios como se muestra a continuación, como criterio de terminación se utiliza un número de iteraciones o cuando se encuentra una solución factible.

Vecindario mover evento

El primer vecindario consiste, en escoger aleatoriamente a algún evento que ya se encuentra asignado, y se pueda mover a otro espacio-tiempo. Como se muestra en la figura 4.8, se elige el evento 22, la asignación cambia de 13 a 78. Y se verifica que los eventos que no han sido asignados tengan una posibilidad de ser asignados.

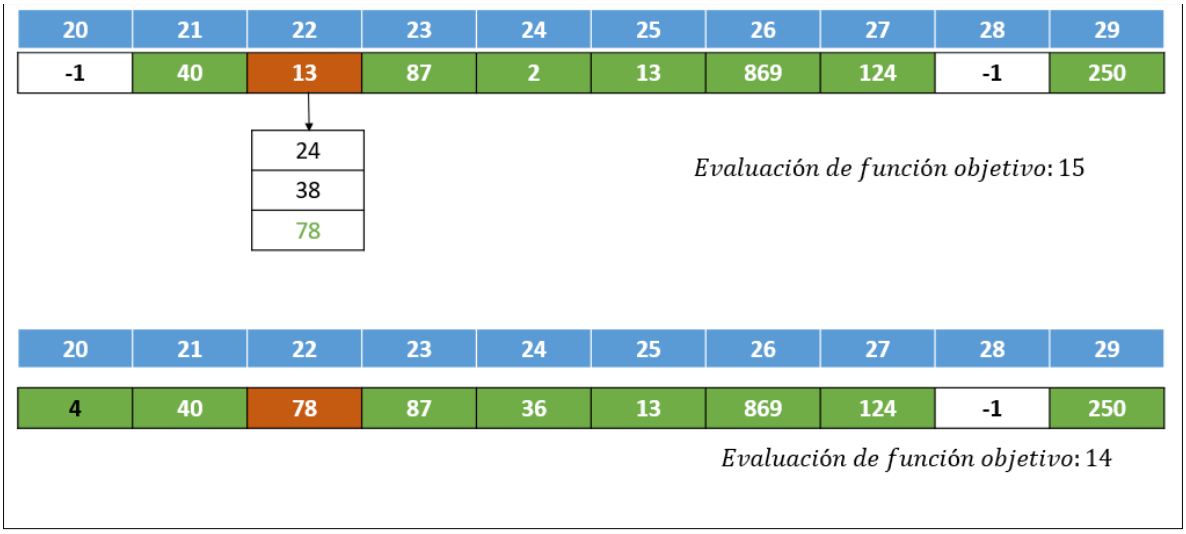

Figura 4.8 Vecindario mover

Vecindario doble mover evento

Este vecindario consiste, en seleccionar dos eventos aleatoriamente que ya se encuentras asignados y moverlos a otro espacio-tiempo. Como se muestra en la figura 4,9, se seleccionaron dos eventos, el evento 21 y 24, cambiándolos de asignación, y así generando una nueva solución.

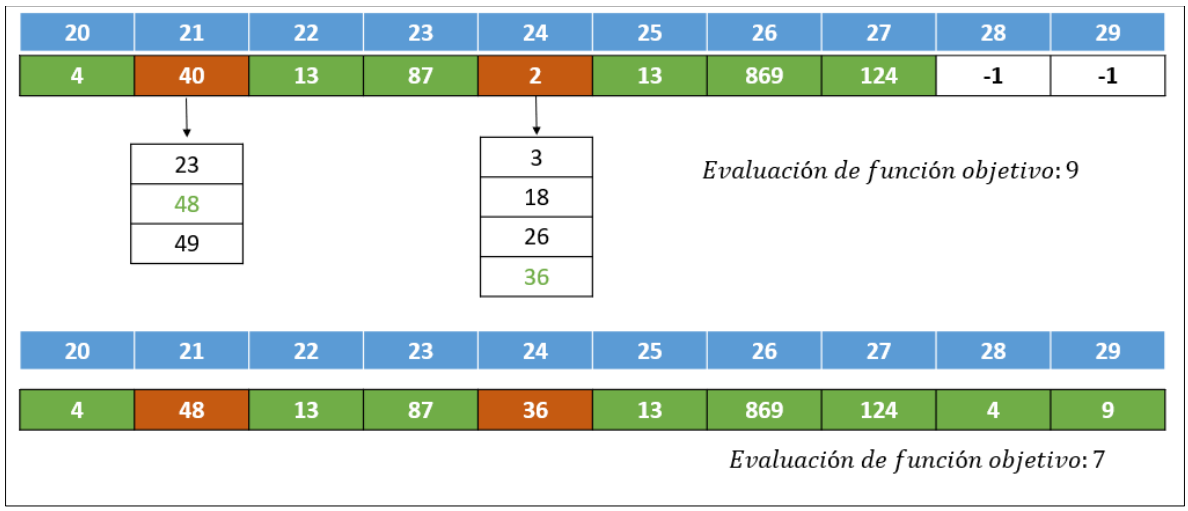

Figura 4.9 Vecindario doble

Vecindario mover todos los eventos

Por último, el vecindario diseñado consiste en mover todos los eventos que tienen posibilidad de moverse a un nuevo horario-aula factible, en cada iteración se mueven todos los eventos a nuevas asignaciones factibles, y después de la iteración se busca que las clases que no han sido asignadas encuentren un aula-horario factible como se muestra en la figura 4.10.

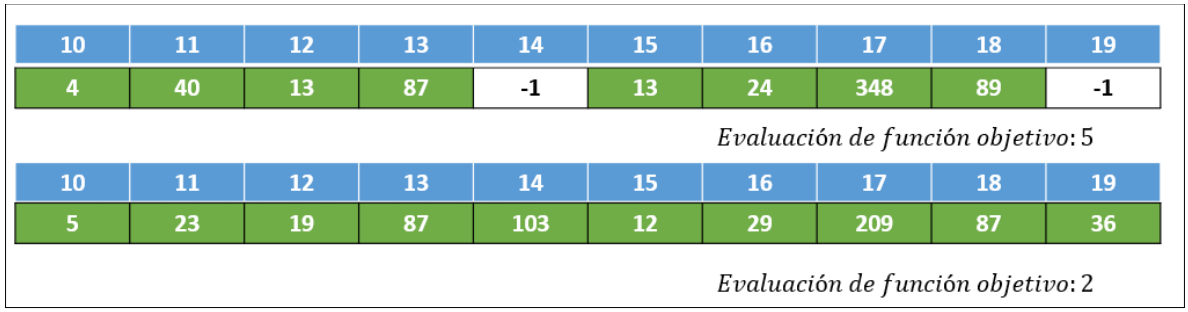

Figura 4.10 Vecindario mover eventos

4.3.2. Pseudocódigo algoritmo de búsqueda local

En la siguiente figura se puede apreciar el pseudocódigo del algoritmo de búsqueda local.

```
1   Algoritmo BusquedaLocal
2   Leer_archivo()
3   Generar_solucion_inicial()
4   Costo_inicial=cuenta_clases_sin_asignar()
5   Mejor_costo=Costo_inicial
6   Iter=0
7       Mientras Iter<MAXITER Hacer
8           Si Mejor_costo == 0 entonces
9               Romper_ciclo
10          Fin si
11          Buscar_otra_solucion()
12          Verificar_factibilidad()
13          Costo_nuevo = cuenta_clases_sin_asignar()
14          Costo_diferencia = costo_nuevo - costo_incial
15          Si costo_diferencia<=0 Entonces
16              Costo_inicial = costo_nuevo
17              Mejor_costo = costo_nuevo
18              Iter=0
19          Fin Si
20          Iter++
21      Fin Mientras
22  Imprimir mejor_solucion
23  Fin Algoritmo
```

Figura 4.11Pseudocódigo del algoritmo de búsqueda local

Para el algoritmo de búsqueda local, se genera una solución inicial con el algoritmo descrito anteriormente, el costo inicial es el número de clases que no han sido asignadas hasta el momento. El número de maxIter se sintonizó como se presenta en la siguiente sección, para el algoritmo. La función buscar_otra_solución() consiste en aplicar el vecindario descrito anteriormente. La variable costo_inicial siempre almacena la mejor solución encontrada por

el algoritmo, cuando esta variable equivale a cero, quiere decir que se encontró una solución factible.

4.3.3. Sintonización de parámetros para búsqueda local

Las pruebas para la sintonización del algoritmo de búsqueda local fueron ejecutadas en una computadora con procesador Intel Pentium J3710, la velocidad del CPU era de 1.60 GHz, 8GB RAM, sistema operativo Windows 10.

A continuación, se presenta la sintonización del parámetro **MAXITER**, el programa se ejecutó 30 veces, y se obtuvo el porcentaje de soluciones factibles. Para la sintonización del parámetro MAXITER se utilizaron dos instancias, muni-pdf-spr16, la muni-fspsx-fal17 y mary-spr17. Se presentan los resultados de los vecindarios, en la tabla 4.1 y 4.2 se puede apreciar la sintonización de MAXITER, en primera columna se aprecia el número de iteraciones, en las columnas posteriores, se aprecia el porcentaje de factibilidad encontrado para cada instancia, dentro de las 30 iteraciones.

Primer vecindario

Tabla 4.1 Sintonización del primer vecindario

MAXITER	muni-pdf-spr16	muni-fspsx-fal17	mary-spr17
1,000	20%	17%	7%
2,000	23%	20%	20%
5,000	37%	37%	47%
10,000	57%	60%	73%
15,000	57%	67%	73%
20,000	57%	67%	73%

Segundo vecindario

Tabla 4.2 Sintonización del segundo vecindario

MAXITER	muni-pdf-spr16	muni-fspsx-fal17	mary-spr17
1,000	23%	20%	10%
2,000	23%	27%	23%
5,000	43%	33%	77%
10,000	67%	67%	83%
15,000	73%	67%	83%
20,000	73%	67%	83%

Tercer vecindario

A continuación, se presentan los resultados con el vecindario, que consiste en mover toda la solución. Cabe destacar que aquí el número de MAXITER es menor debido, a que requiere más trabajo de cómputo realizar el vecindario, por lo tanto, se decidió bajar el número de iteraciones.

Tabla 4.3 Sintonización del tercer vecindario

MAXITER	muni-pdf-spr16	muni-fspsx-fal17	mary-spr17
30	43%	37%	63%
40	43%	43%	73%
50	47%	47%	90%
60	60%	47%	90%
80	73%	63%	93%
100	80%	67%	93%
120	87%	73%	93%
140	87%	73%	93%

Hibridación de los tres

Por último, se optó, por implementar los tres vecindarios propuestos en el mismo algoritmo, dando 33.33 de probabilidad a cada vecindario. La implementación de este método fue el mejor para ambas instancias.

Tabla 4.4 Sintonización del vecindario hibrido

MAXITER	muni-pdf-spr16	muni-fspsx-fal17	mary-spr17
250	77%	60%	83%
500	90%	67%	90%
750	93%	87%	90%
1,000	93%	90%	93%
1250	93%	90%	93%
1500	93%	90%	93%

Después de sintonizar los vecindarios, se determinó que el mejor vecindario es el de hibridación con MAXITER igual a mil.

4.4. Algoritmo para asignar estudiantes

Después de encontrar un horario factible para todas las clases, es necesario asignar a los estudiantes, de tal forma que asistan a todos los cursos solicitados. Se va seleccionando estudiante por estudiante. Después de seleccionar al estudiante, se selecciona el primer curso al que quiere asistir, se busca en la primera configuración del curso si existen lugares disponibles si es así, se asigna a esa configuración al estudiante, de no existir lugares, se busca en otra configuración del curso, así hasta encontrar una configuración factible es decir que exista cupo y así poder asignar al estudiante, como se muestra en la figura 4.12. El algoritmo no verifica que el estudiante sea asignado a dos clases cuyos horarios choquen entre sí, dado que en este problema esto se considera una restricción suave.

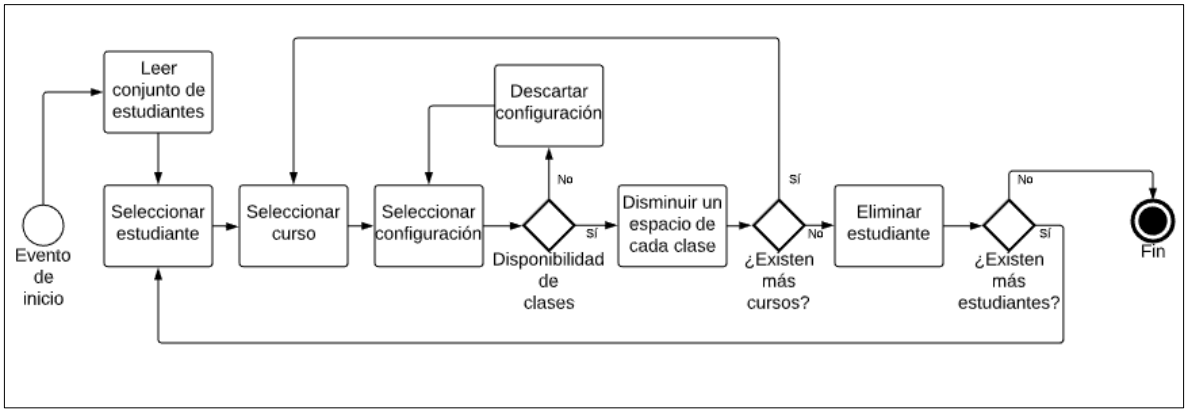

Figura 4.12 Diagrama del algoritmo para asignar estudiantes

4.5. Resultados

Durante la competencia, se logró subir a tiempo seis soluciones factibles, son las que se muestra en la tabla 4.5, se puede apreciar el costo, y el mejor costo que se obtuvo de distintos autores. Con los resultados obtenidos se logró estar en la posición número 9 de los resultados, como se muestra en la figura 4.13, cabe destacar que los resultados se están actualizando constantemente en la página oficial de ITC-2019. En esta etapa de la competencia solo se había implementado el algoritmo constructivo y la búsqueda local con el primer vecindario sin la sintonización de parámetros. Resultados durante la competencia

Tabla 4.5 Resultados obtenidos durante el lapso de la competencia

Instancia	Costo	Mejor costo
lums-spr18	1509	95
muni-fsps-spr17c	608794	2596
muni-pdf-spr16	352344	17208
lums-fal17	2720	349
mary-fal18	59466	4422
muni-fi-fal17	25451	2999

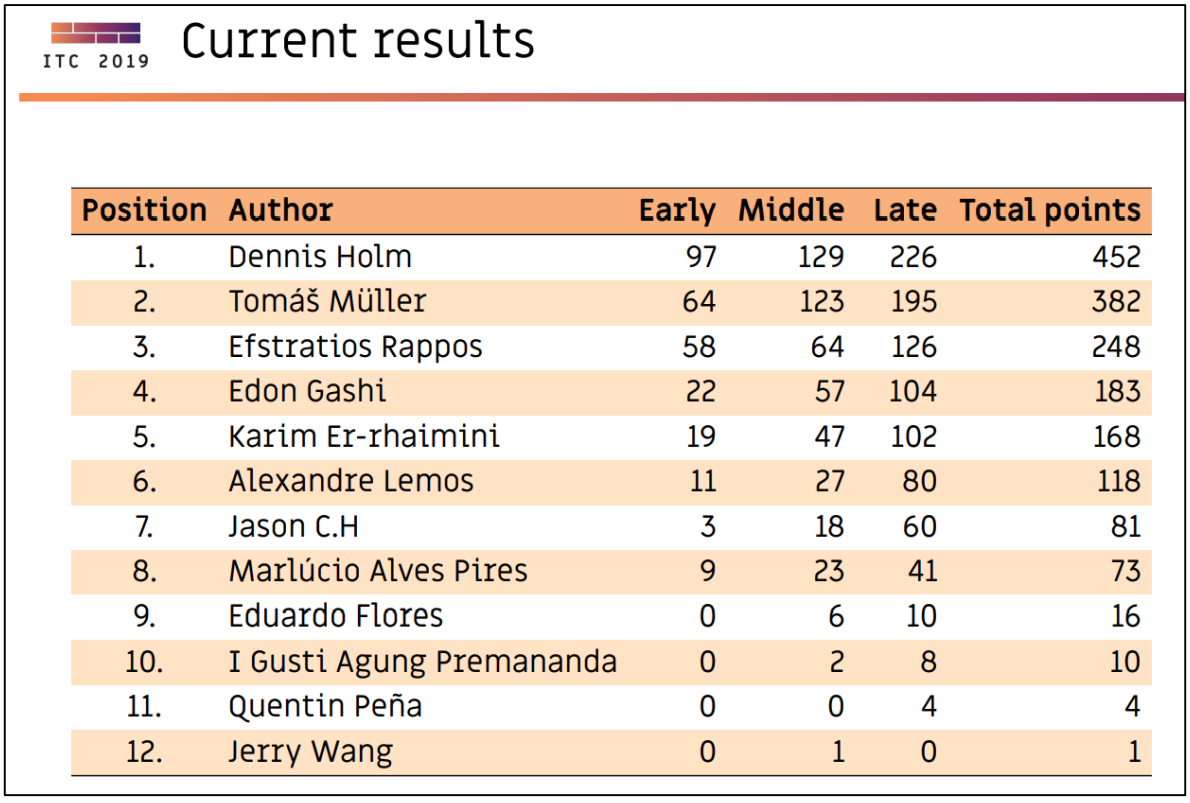

Figura 4.13 Ranking de resultados de la competencia ITC 2019

Para calcular los puntos obtenidos de cada participante se estableció una tabla que indica la cantidad de puntos por cada instancia, dependiendo del lugar que haya obtenido cada participante. Por ejemplo, si obtienes el lugar 1° en alguna instancia “Early”, obtendrás 10 puntos, y así se sumarán todos los puntos obtenidos en cada instancia para calcular el total.

Tabla 4.6. Ponderación de puntos para las instancias

	Instancia		
Posición	**Early**	**Middle**	**Late**
1°	10	15	24
2°	7	11	18
3°	5	8	15
4°	3	6	12
5°	2	4	10
6°	1	3	8
7°		2	6
8°		1	4
9°			2
10°			1

Resultados Finales

Aquí se presentan los resultados finales obtenidos con el algoritmo propuesto, ya con la sintonización de los parámetros, se lograron obtener 12 soluciones factibles, también se logró resolver todas las instancias de juguete proporcionadas por la competencia como se muestra en la tabla 4-7, mientras que la tabla 4-8 muestra las soluciones factibles obtenidas, así como el costo, y el tiempo de ejecución. También se muestra el mejor costo encontrado por otros autores, así como su tiempo de solución en segundos.

Tabla 4-7. Resultados de instancias de prueba

Instancia	Costo Total	Tiempo(s)
wbg-fal10	914	0
lums-sum17	43	12
bet-sum18	3453	12
pu-cs-fal07	2060	0
pu-llr-spr07	90,456	19

Cabe destacar que existe gran diferencia entre el costo obtenido y el de los otros autores, debido a que esta tesis se enfocó en encontrar factibilidad, y no en la parte de optimización, como se aprecia en la tabla 4-8.

Tabla 4-8. Resultados obtenidos

Instancia	Costo	Tiempo (s)	Mejor Costo	Tiempo (s)
mary-spr17	70,422	10	14,910	1,069
muni-fi-spr16	26,752	2	3,756	311,147
muni-fsps-spr17	222,606	8	868	864,000
pu-llr-spr17	128,302	8	10,038	864,000
lums-spr18	1,402	12	95	17,374
muni-fi-spr17	24,941	3	3,825	771,543
muni-fsps-spr17c	502,794	26	2,596	918,446
muni-pdf-spr16	332,344	48	17,208	476,372
lums-fal17	2,240	12	349	67,129
mary-fal18	59,466	3	4,422	199,895
muni-fi-fal17	25,451	5	2,999	230,359
muni-fspsx-fal17	143,302	59	10,123	18,340

Para calcular el costo total de cada instancia, varía según el criterio de cada institución, por eso al momento de leer la instancia, vienen los pesos de cada criterio para así poder calcular el costo total, a continuación, se presenta un ejemplo con la instancia mary-spr17:

Tabla 4-9. Explicación detallada del cálculo de costo total

Tipo	**Peso**	**Penalización**	**Total**
Tiempos	2	9429	18858
Aulas	1	4029	4029
Restricciones	5	6903	34515
Estudiantes	10	1302	13020
Costo total			**70422**

Como se puede observar se multiplica el peso que se la da al criterio por la penalización de este, y al final se suman todos los criterios, así obteniendo el costo total. En la tabla 4.10 se presentan los resultados obtenidos para las instancias resueltas.

Tabla 4-10. Resultados con penalizaciones

Instancia	**Costo Total**	**Conflictos de estudiantes**	**Penalización de tiempo**	**Penalización de aula**	**Penalización de distribución**
wbg-fal10	914	53	192	42	15
lums-sum17	43	0	20	23	0
bet-sum18	3453	0	12	2481	96
pu-cs-fal07	2060	51	546	304	70
pu-llr-spr07	90456	7010	13345	6451	56
mary-spr17	70,422	9429	4029	6903	1302
muni-fi-spr16	26,752	3344	2168	2488	104
muni-fsps-spr17	222,606	2118	210	366	346
pu-llr-spr17	128,302	7855	15455	15081	251
lums-spr18	1,402	0	204	711	78
muni-fi-spr17	24,941	3453	1714	2044	49
muni-fsps-spr17c	502,794	5923	514	1244	160
muni-pdf-spr16	332,344	7653	51444	35196	813
lums-fal17	2,240	0	522	826	170
mary-fal18	59,466	3942	7095	4436	284
muni-fi-fal17	25,451	3234	1848	1527	221
muni-fspsx-fal17	143,3024	3456	26487	14195	1243

5. Conclusiones y trabajo futuro

En este capítulo se abordan las conclusiones y trabajo futuro para esta tesis de investigación.

5.1. Conclusiones

Se implementó una heurística constructiva, que va asignando clases una por una, según un criterio de restricción, la cual genera la solución inicial, después se aplica un algoritmo de búsqueda local que tiene la estrategia de minimizar el número de clases sin asignar para poder llegar a una solución factible.

También se implementaron 3 vecindarios diferentes: simple, doble y mover todas las clases posibles para el problema del ITC 2019. Como se mostró en la parte de sintonización, el porcentaje de factibilidad más alto se encontró en la hibridación de los tres vecindarios con 1000 iteraciones sin mejorar de la mejor solución global.

Se logró efectuar un algoritmo que utiliza 3 vecindarios hibridados para resolver el complejo problema de ITC2019, en el algoritmo se verifica la factibilidad cada vez que se efectúa un movimiento. Se diseñó un algoritmo para asignar a los estudiantes una vez que todas las clases se encuentran en asignaciones factibles. Con las heurísticas implementadas se logró encontrar 12 soluciones factibles para las instancias de la competencia ITC2019. El enfoque presentado de hibridar tres vecindarios tiende a generar soluciones factibles, por lo que podemos intuir que si se generaran más vecindarios y se implementaran metaheurísticas se podrían encontrar más soluciones factibles. A pesar de que el trabajo de esta tesis no minimiza la violación de las restricciones suaves, se logró estar en el top 10, específicamente en la posición 9 de los resultados publicados por la competencia ITC2019.

5.2. Trabajo futuro

A continuación, se enlistan algunas líneas de investigación, que por falta de tiempo y por la complejidad del problema, no pudieron abordarse en el presente trabajo, pero que se podrían abordar en futuras investigaciones:

- Como trabajo futuro se pretende diseñar un vecindario más inteligente, es decir que se enfoque en las clases y cursos que no han podido asignarse y en las asignaciones de otras clases que impiden que las primeras no puedan asignarse por estar utilizando recursos que necesitan para ser programadas.
- Mejorar el algoritmo para asignar estudiantes tratando de minimizar los conflictos de estudiantes, es decir que disminuya el traslape de clases asignadas a los estudiantes
- Implementar las metaheurísticas de Recocido Simulado, Búsqueda Tabú, o alguna otra metaheurística para resolver el problema.
- Permitir soluciones infactibles, generando soluciones aleatorias y dándoles un peso significativo a las restricciones duras violadas, y así tratar de minimizar la función objetivo, tratando de que la solución final sea factible.
- Abordar el problema en la parte de optimización (horarios, aulas, restricciones y estudiantes) para minimizar el peso de las restricciones suaves violadas.

Referencias

Abdullah, S., & Turabieh, H. (2008). Generating university course timetable using genetic algorithms and local search. *Third International Conference on Convergence and Hybrid Information Technology*, 254-260.

Abdullah, S., Burke, E., & McCollum, B. (2007). A hybrid evolutionary approach to the university course timetabling problem. *IEEE congress on evolutionary computation*, 1764-1768.

Aho, A. (1974). *THE DESIGN AND ANALYSIS OF COMPUTER ALGORITHMS.* London: Addison-W~ley Publishing Company.

Al-Betar, M., & Khader, A. (2012). A harmony search algorithm for university course timetabling. *Annals of Operations Research*, 3-31.

Alonso, F. (2008). *Programación de horarios escolares con recocido simulado y búsqueda tabú.* Cuernavaca.

Babaei, H., Karimpour, J., & Hadidi, A. (2015). A survey of approaches for university course timetabling problem. *Computers & Industrial Engineering*, 43-59.

Basir, N., Ismail, W., & Norwawi, N. (2013). A simulated annealing for Tahmidi course timetabling. *Procedia Technology*, 437-445.

Burke, E., Jackson , K., & Kingston, K. (1997). Automated university timetabling: The state of the art. *The computer journal*, 565-571.

Burke, E., Marecek, J., Parkes, A., & Rudová, H. (2012). A branch-and-cut procedure for the Udine course timetabling problem. *Annals of Operations Research*, 71-87.

Cambazard, H., & Hebrard, E. (2012). Local search and constraint programming for the post enrolment-based course timetabling problem. *Local search and constraint programming for the post enrolment-based course timetabling problem*, 111-135.

Carter, M. (2000). A comprehensive course timetabling and student scheduling system at the University of Waterloo. *In International Conference on the Practice and Theory of Automated Timetabling*, 64-82.

Ceschia, S., Di Gaspero, L., & Schaerf, A. (2012). Design, engineering, and experimental analysis of a simulated annealing approach to the post-enrolment course timetabling problem. *Computers & Operations Research*, 1615-1624.

de Werra, D. (1985). An introduction to timetabling. *European journal of operational research*, 151-162.

Frausto Solis, J., Alonso Pecina, F., Larre, M., GONZALEZ SEGURA, C., & GÓMEZ RAMOS, J. (2006). Solving the timetabling problem with three heuristics. *small*, 100(5), 5.

Geiger, M. (2012). Applying the threshold accepting metaheuristic to curriculum based course timetabling. *Annals of Operations Research*, 189-202.

Lü, Z., & Hao, J. (2010). Adaptive tabu search for course timetabling. *European journal of operational research*, 235-244.

Müller, T. (2009). ITC2007 solver description: a hybrid approach. *Springer Science+Business Media*, 429–446.

Nothegger, C., Mayer, A., Chwatal, A., & Raidl, G. (2012). Solving the post enrolment course timetabling problem by ant colony optimization. *Annals of Operations Research*, 325-339.

Phillips, A., Walker, C., Ehrgott, M., & Ryan, D. (2017). Integer programming for minimal perturbation problems in university course timetabling. *Annals of Operations Research*, 283-304.

Sapru, V., Reddy, K., & Sivaselvan, B. (2010). Time Table Scheduling using Genetic Algorithms. *IEEE*, 1-4.

Schaerf, A. (1999). A survey of automated timetabling. *Artificial intelligence review*, 87-127.

Junn, K. Y., Obit, J. H., & Alfred, R. (2016). Comparison of simulated annealing and great deluge algorithms for university course timetabling problems (UCTP). Advanced Science Letters, 23(11), 11413-11417

Holm, D. S., Mikkelsen, R. Ø., Sørensen, M., & Stidsen, T. J. R. (2020). A MIP Formulation of the International Timetabling Competition 2019 Problem.

Cruz-Chávez, M. A., Flores-Pichardo, M., Martínez-Oropeza, A., Moreno-Bernal, P., & Cruz-Rosales, M. H. (2016). Solving a Real Constraint Satisfaction Model for the University Course Timetabling Problem: A Case Study. Mathematical Problems in Engineering, 2016.

Borchani, R., Elloumi, A., & Masmoudi, M. (2017). Variable neighborhood descent search based algorithms for course timetabling problem: Application to a Tunisian University. Electronic Notes in Discrete Mathematics, 58, 119-126.

Goh, S. L., Kendall, G., & Sabar, N. R. (2017). Improved local search approaches to solve the post enrolment course timetabling problem. European Journal of Operational Research, 261(1), 17-29.

Junn, K. Y., Obit, J. H., & Alfred, R. (2017). A constraint programming approach to solving university course timetabling problem (UCTP). Advanced Science Letters, 23(11), 11023-11026.

Kralev, V., & Kraleva, R. (2017). A local search algorithm based on chromatic classes for university course timetabling problem. International Journal of Advanced Computer Research, 7(28).

Rezaeipanah, A., Matoori, S. S., & Ahmadi, G. (2020). A hybrid algorithm for the university course timetabling problem using the improved parallel genetic algorithm and local search. Applied Intelligence, 1-26.

Thepphakorn, T., & Pongcharoen, P. (2020). Performance improvement strategies on Cuckoo Search algorithms for solving the university course timetabling problem. Expert Systems with Applications, 161, 113732.

Abdelhalim, E. A., & El Khayat, G. A. (2016). A utilization-based genetic algorithm for solving the university timetabling problem (uga). Alexandria Engineering Journal, 55(2), 1395-1409

Pereira, V., & Gomes Costa, H. (2016). Linear integer model for the course timetabling problem of a faculty in Rio de Janeiro. Advances in Operations Research, 2016.

Zheng, S., Wang, L., Liu, Y., & Zhang, R. (2015). A simulated annealing algorithm for university course timetabling considering travelling distances. International Journal of Computing Science and Mathematics, 6(2), 139-151.

Gülcü, A., & Akkan, C. (2020). Robust university course timetabling problem subject to single and multiple disruptions. European Journal of Operational Research, 283(2), 630-646.

Wang, K., Shang, W., Liu, M., Lin, W., & Fu, H. (2018, June). A Greedy and Genetic Fusion Algorithm for Solving Course Timetabling Problem. In 2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS) (pp. 344-349). IEEE.

Kenekayoro, P., & Zipamone, G. (2016). Greedy ants colony optimization strategy for solving the curriculum based university course timetabling problem. arXiv preprint arXiv:1602.04933

Mühlenthaler, M., & Wanka, R. (2016). Fairness in academic course timetabling. Annals of Operations Research, 239(1), 171-188.

Francis, K., Manga, I., & Sarjiyus, O. (2016). Scheduling algorithm for university timetabling problem. IOSR J. Comput. Eng, 18(6), 39-43.

Prabodanie, R. R. (2017). An integer programming model for a complex university timetabling problem: a case study. Industrial Engineering & Management Systems, 16(1), 141-153

Soria-Alcaraz, J. A., Özcan, E., Swan, J., Kendall, G., & Carpio, M. (2016). Iterated local search using an add and delete hyper-heuristic for university course timetabling. Applied Soft Computing, 40, 581-593.

Goh, S. L., Kendall, G., Sabar, N. R., & Abdullah, S. (2020). An effective hybrid local search approach for the post enrolment course timetabling problem. Opsearch, 57(4), 1131-1163.

Sze, S. N., Bong, C. L., Chiew, K. L., Tiong, W. K., & Bolhassan, N. A. (2017, May). Case study: University lecture timetabling without pre-registration data. In 2017 International Conference on Applied System Innovation (ICASI) (pp. 732-735). IEEE.

Song, T., Liu, S., Tang, C., Peng, X., & Chen, M. (2018). An iterated local search algorithm for the University Course Timetabling Problem. *Applied Soft Computing*, 597

Cooper, T. B., & Kingston, J. H. (1995, August). The complexity of timetable construction problems. In International conference on the practice and theory of automated timetabling (pp. 281-295). Springer, Berlin, Heidelberg.

Apéndice A

En la figura A 1, viene el formato XML, de cómo sería la lectura de las aulas, donde cada aula tiene su identificador, capacidad, la distancia entre aulas, y también como se puede ver en el aula con el identificador tres, nos muestra las semanas y días que no está disponible el aula, por lo tanto, no se puede ocupar ese horario para colocar una clase.

```
<rooms>
  <room id="1" capacity="50"/>
  <room id="2" capacity="100">
    <travel room="1" value="2"/> <!-- travel time is in the number of slots -->
  </room>
  <room id="3" capacity="80">
    <travel room="2" value="3"/> <!-- only non-zero travel times are present -->
    <!-- not available on Mondays and Tuesdays betwen 8:30 - 10:30, all weeks -->
    <unavailable days="110000" start="102" length="24" weeks="111111111111"/>
    <!-- not available on Fridays betwen 12:00 - 24:00, odd weeks only -->
    <unavailable days="000100" start="144" length="144" weeks="101010101010"/>
  </room>
</rooms>
```

Figura A 1. Ejemplo de aulas en código de lenguaje

En la figura A 2, viene un ejemplo, donde la clase con identificador uno, tiene una capacidad de veinte estudiantes y tiene la posibilidad de ser asignada en dos aulas, cada aula con una penalización específica, por otro lado, también pueden existir clases sin aula, como por ejemplo la clase con identificador 2, nos indica que no requiere aula, pero si tiene una capacidad de diez estudiantes.

```
<class id="1" limit="20">
  <room id="1" penalty="20"/>
  <room id="3" penalty="0"/>
</class>
<class id="2" limit="10" rooms="false"/>
```

Figura A 2. Ejemplo de aulas disponibles para una aula

Como se puede observar en la figura A 3, donde cada estudiante tiene una lista de cursos, a los cuales quiere asistir. Por lo tanto, se tiene como objetivo, asignar de manera correcta los cursos solicitados por los estudiantes

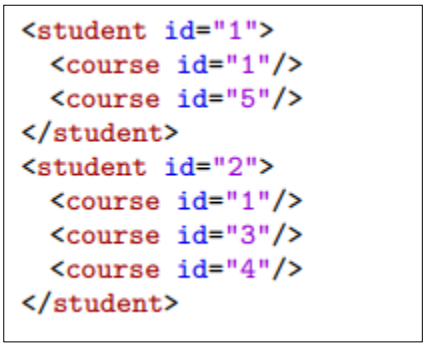

Figura A 3. Demanda de cursos por parte de los estudiantes en código de lenguaje.

Apéndice B

En este apéndice se describen ejemplos de las restricciones.

En la figura B1, se puede observar cómo están estructuradas las restricciones, por ejemplo, en la primera, de tipo “NotOverlap” nos indica que es una restricción dura, que se debe satisfacer, de tal manera que la clase 1 y 2 deben cumplir con “notOverlap”. En la segunda restricción se puede apreciar que es una restricción de tipo suave, y viene asociada una penalización de valor 2.

```
<distributions>
  <!-- classes 1 and 2 cannot overlap in time -->
  <distribution type="NotOverlap" required="true">
    <class id="1"/>
    <class id="2"/>
  </distribution>
  <!-- class 1 should be before class 3, class 3 before class 5 -->
  <distribution type="Precedence" penalty="2">
    <class id="1"/>
    <class id="3"/>
    <class id="5"/>
  </distribution>
</distributions>
```

Figura B 1. Especificación de las restricciones de distribución en código de lenguaje

En la figura B2 indica que la clase 1109 y 1110 deben cumplir con la restricción SameStart, como se aprecia las clases han sido asignadas con el inicio de 114, y así se cumple con la restricción.

```
<solution>
        <distribution type="SameStart" required="true">
                <class id="1109"/>
                <class id="1110"/>
        </distribution>
        <class id="1109" days="1010000" start="114" weeks="111111111111111" room="3"></class>
        <class id="1110" days="0000100" start="114" weeks="111111111111111" room="3"></class>
</solution>
```

Figura B 2. Restricción SameStart en código de lenguaje

En la figura B3, se da un ejemplo de la restricción SameTime.

```
<solution>
        <distribution type="SameTime" required="true">
                <class id="165"/>
                <class id="163"/>
        </distribution>
        <class id="163" days="0100000" start="162" weeks="111111111111111" room="157"></class>
        <class id="165" days="0001000" start="162" weeks="111111111111111" room="157"></class>
</solution>
```

Figura B 3. Restricción SameTime en código de lenguaje

En la figura B4, se da un ejemplo de la restricción DifferentTime.

```
<solution>
        <distribution type="DifferentTime" required="true">
                <class id="219"/>
                <class id="241"/>
        </distribution>
        <class id="219" days="0101000" start="186" weeks="111111111111111" room="28"></class>
        <class id="241" days="1010100" start="126" weeks="111111111111111" room="25"></class>
</solution>
```

Figura B 4. Restricción DifferentTime en código de lenguaje

Por ejemplo, si tenemos tres clases, y una fue asignada al lunes, martes y miércoles, las otras dos restantes, solo pueden ser asignadas a esos días. Como se puede observar en la siguiente figura B5, se muestra la restricción sameDays con tres clases.

```
<solution>
        <distribution type="SameDays" required="true">
                <class id="1477"/>
                <class id="1478"/>
                <class id="1479"/>
        </distribution>
        <class id="1477" days="1110000" start="172" weeks="111111111111111" room="21"></class>
        <class id="1478" days="0110000" start="202" weeks="111111111111111" room="8"></class>
        <class id="1479" days="0010000" start="196" weeks="111111111111111" room="1"></class>
</solution>
```

Figura B 5. Restricción SameDays en código de lenguaje

Como se puede apreciar en la figura B6, las tres clases deben ser programadas de tal manera que no compartan ningún día

```
<solution>
        <distribution type="DifferentDays" required="true">
                <class id="2340"/>
                <class id="2341"/>
                <class id="2342"/>
        </distribution>
        <class id="2340" days="0001000" start="114" weeks="111111111111111" room="138"></class>
        <class id="2341" days="0100000" start="162" weeks="111111111111111" room="173"></class>
        <class id="2342" days="0010000" start="162" weeks="111111111111111" room="173"></class>
</solution>
```

Figura B 6. Restricción DifferentDays en código de lenguaje.

De igual forma que la restricción SameDays, la restricción es igual, pero con las semanas.

```
 <solution>
        <distribution type="SameWeeks" required="true">
                <class id="2343"/>
                <class id="2344"/>
        </distribution>
        <class id="2343" days="0100000" start="186" weeks="111111111111111" room="138"></class>
        <class id="2344" days="0100000" start="186" weeks="111111111111111" room="172"></class>
 </solution>
```

Figura B 7. Restricción SameWeeks en código de lenguaje.

```
<solution>
        <distribution type="DifferentWeeks" required="true">
                <class id="2343"/>
                <class id="2344"/>
        </distribution>
        <class id="12" days="0100000" start="196" weeks="010101010101010" room="24"></class>
        <class id="27" days="0100000" start="176" weeks="101010101010101" room="24"></class>
</solution>
```

Figura B 8. Restricción DifferentWeeks en código de lenguaje

A continuación, se presenta un ejemplo de la restricción overlap.

```
<solution>
        <distribution type="Overlap" required="true">
                <class id="848"/>
                <class id="850"/>
        </distribution>
        <class id="848" days="0001000" start="186" weeks="110000000000000" room="122"></class>
        <class id="850" days="0001000" start="186" weeks="110000000000000" room="58"></class>
</solution>
```

Figura B 9. Restricción Overlap en código de lenguaje

En el siguiente ejemplo figura B 10, se supone que las cuatro clases comparten al menos un día a la semana, y una semana durante el semestre, por lo tanto, el inicio y final de cada clase, debe cumplir con la restricción anterior.

```
<solution>
        <distribution type="NotOverlap" required="true">
                <class id="865"/>
                <class id="866"/>
                <class id="871"/>
                <class id="872"/>
        </distribution>
        <class id="865" days="0001000" start="186" weeks="100000000000000" room="122"></class>
        <class id="866" days="0001000" start="198" weeks="100000000000000" room="122"></class>
        <class id="871" days="0010100" start="138" weeks="111111111111111" room="34"></class>
        <class id="872" days="1000000" start="114" weeks="111111111111111" room="118"></class>
</solution>
```

Figura B 10. Restricción NotOverlap en código de lenguaje

La restricción SameRoom, dice que las clases deben ser asignadas a la misma aula, como se aprecia en la figura B11.

```
<solution>
        <distribution type="SameRoom" required="true">
                <class id="1370"/>
                <class id="1371"/>
        </distribution>
        <class id="1370" days="1010100" start="114" weeks="111111111111111" room="102"></class>
        <class id="1371" days="1010100" start="102" weeks="111111111111111" room="102"></class>
</solution>
```

Figura B 11. Restricción SameRoom en código de lenguaje

A continuación, su contraparte de la restricción SameRoom, en la figura B12.

```
<solution>
        <distribution type="DifferentRoom" required="true">
                <class id="5"/>
                <class id="8"/>
                <class id="9"/>
        </distribution>
        <class id="5" days="1010100" start="102" weeks="111111111111111" room="2"></class>
        <class id="8" days="1010100" start="96" weeks="111111111111111" room="48"></class>
        <class id="9" days="1010100" start="72" weeks="111111111111111" room="36"></class>
</solution>
```

Figura B 12. Restricción DifferentRoom en código de lenguaje

La restricción SameAttendees dice que debemos asignar las clases, de tal forma que le demos tiempo al estudiante o profesor de poder llegar a las clases, sin que exista traslape y la distancia entre aulas sea un impedimento. Como se puede apreciar en la figura B13.

```
<solution>
        <distribution type="SameAttendees" required="true">
                <class id="35"/>
                <class id="28"/>
                <class id="21"/>
        </distribution>
        <room id="147" capacity="21">
                <travel room="209" value="2"/>
        </room>
        <class id="35" days="1000000" start="114" weeks="111111111111111" room="147"></class>
        <class id="28" days="0010000" start="114" weeks="111111111111111" room="147"></class>
        <class id="21" days="1000000" start="186" weeks="111111111111111" room="209"></class>
</solution>
```

Figura B 13. Restricción SameAttendees en código de lenguaje

En la figura 3.20, se puede apreciar que todas las clases cumplen con la restricción de precedencia.

```
<solution>
        <distribution type="Precedence" required="true">
                <class id="1024"/>
                <class id="1025"/>
                <class id="1026"/>
                <class id="1027"/>
                <class id="1028"/>
        </distribution>
        <class id="1024" days="0100000" start="174" weeks="111111111111111" room="93"></class>
        <class id="1025" days="0010000" start="126" weeks="111111111111111" room="93"></class>
        <class id="1026" days="0000100" start="90" weeks="111111111111111" room="93"></class>
        <class id="1027" days="0000100" start="126" weeks="111111111111111" room="93"></class>
        <class id="1028" days="0000100" start="180" weeks="111111111111111" room="93"></class>
</solution>
```

Figura B 14. Restricción Precedence en código de lenguaje

Como se puede observar en la figura B15 el valor de S es igual a 24, ambas clases comparten semanas y días, por lo tanto, se debe validar que el final de la última clase 124, y el inicio de la primera clase 102, debe cumplir que $124 - 102 \leq 24 \Longrightarrow 22 \leq 24$ por lo tanto se cumple con la restricción.

```
<solution>
        <distribution type="WorkDay(24)" required="true">
                <class id="1370"/>
                <class id="1371"/>
        </distribution>
        <class id="1370" days="1010100" start="114" end="124" weeks="111111111111111" room="102"></class>
        <class id="1371" days="1010100" start="102" end="112" weeks="111111111111111" room="102"></class>
</solution>
```

Figura B 15. Restricción WorkDay en código de lenguaje

En la figura B16, el valor de G nos indica que es 6, por lo tanto, ambas clases deben estar separadas al menos g periodos de tiempo, como ambas clases se traslapan en días y semanas, se debe validar lo siguiente: $136 + 6 \leq 186 \Longrightarrow 142 \leq 186$, por lo tanto, se cumple con la restricción.

```
<solution>
        <distribution type="MinGap(6)" required="true">
                <class id="1712"/>
                <class id="1790"/>
        </distribution>
        <class id="1712" days="1010100" start="126" end="136" weeks="111111111111111" room="62"></class>
        <class id="1790" days="1010100" start="186" end="196" weeks="111111111111111" room="77"></class>
</solution>
```

Figura B 16. Restricción MinGap en código de lenguaje

En la figura B17. Tenemos tres clases que nos indica que el valor de D=2, por lo tanto, las clases no se pueden extender a más de dos días en la semana.

```
<solution>
        <distribution type="MaxDays(2)" required="true">
                <class id="176"/>
                <class id="177"/>
                <class id="178"/>
        </distribution>
</solution>
```

Figura B 17. Restricción MaxDays en código de lenguaje.

En la tabla B1, se presentan los días asignados de las tres clases, como se puede apreciar al final, la función OrFunction, realiza la operación lógica OR de las cadenas de bits que representan los días asignadas de cada clase. Después de obtener la cadena de la función OrFunction se realiza la sumatoria de los bits que tienen valor de uno, como se puede apreciar en la tabla B2, en este caso la sumatoria nos da un dos como valor, y así satisface la restricción.

Tabla B 1. Ejemplo de MaxDays factible 1

Clase	Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
176	1	1	0	0	0	0	0
177	1	1	0	0	0	0	0
178	0	1	0	0	0	0	0
OrFunction	1	1	0	0	0	0	0

Tabla B 2. Ejemplo de OrFunction 1

OrFunction							
Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo	Sumatoria
1	1	0	0	0	0	0	2

De igual manera para la tabla B3, se aplica la función OrFunction, donde la función OrFunction nos devolvería un valor de cuatro, como se aprecia en la tabla B4, en este caso no se cumple con la restricción, lo que estaría generando infactibilidad.

Tabla B 3. Ejemplo de MaxDays infactible 1

Clase	Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
176	1	1	0	0	0	0	0
177	0	0	1	0	0	0	0
178	0	0	0	1	0	0	0
OrFunction	1	1	1	1	0	0	0

Tabla B 4. Ejemplo de OrFunction 2

OrFunction							
Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo	Sumatoria
1	1	1	1	0	0	0	4

En la figura B18, se presenta la restricción de distribución MaxDayLoad para dos clases, con el valor de S igual a 32.

```
<solution>
        <distribution type="MaxDayLoad(32)" required="true">
                <class id="59"/>
                <class id="78"/>
                <class id="85"/>
        </distribution>
</solution>
```

Figura B 18. Restricción MaxDayLoad en código de lenguaje

La función DayLoad suma el tamaño de la clase, cuando está asignada a un día, en el jueves hay dos clases asignadas, por lo tanto, se suma el tamaño de ambas, dando como resultado 32, en este caso se cumple con la restricción debido a que es menor o igual a 32.

Tabla B 5. Ejemplo de MaxDayLoad Factible

	Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Inicio	Final	Tamaño
Clase 58	0	0	0	1	0	0	192	212	20
Clase 78	0	0	0	1	1	0	160	172	12
Clase 85	0	0	1	0	0	0	148	160	12
DayLoad	0	0	12	32	12	0			

En el caso de la tabla B6, se puede observar que las tres clases están asignadas al día jueves, dando una sumatoria del tamaño de clases de 44, por lo tanto, en este ejemplo la solución es infactible.

Tabla B 6. Ejemplo de MaxDayLoad Factible

	Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Inicio	Final	Tamaño
Clase 58	0	0	0	1	0	0	192	212	20
Clase 78	0	0	0	1	1	0	160	172	12
Clase 85	0	0	0	1	0	0	148	160	12
DayLoad	0	0	0	44	12	0			

A continuación, se presenta un ejemplo como se puede observar en la figura b19, la restricción MaxBreaks nos da un valor de R=0 y S=72.

```
<solution>
        <distribution type="MaxBreaks(0,72)" required="true">
                <class id="1023"/>
                <class id="1019"/>
                <class id="1005"/>
        </distribution>
</solution>
```

Figura B 19. Restricción MaxBreaks en código de lenguaje

A continuación, se presenta un ejemplo de esta restricción, siguiendo los datos de la figura B19, se presenta en la tabla B6, el inicio y final de cada clase involucrada en la restricción.

Tabla B 7. Ejemplo de MaxBreaks clases

Clase	**Inicio**	**Final**
1023	**96**	**111**
1019	**114**	**129**
1005	**144**	**154**

De tal forma, que el número de descansos no debe ser mayor a uno, y validando que el inicio y el final entre clases no sea mayor a 72, el resultado es un bloque solamente, como se muestra en la tabla B7, por lo tanto, se cumple con la restricción.

Tabla B 8. Ejemplo de MaxBreaks bloques

Bloque	Inicio	Final	Tamaño
1	96	154	58