# Heurística para la Planificación de Horarios de la Universidad EAFIT


Proyecto de grado presentado
por
John Jairo Silva
a
Departamento de Ciencias Matemáticas


Para optar por el título de
Magister en Matemáticas Aplicadas
en la
Maestría en Matemáticas Aplicadas


Universidad EAFIT
Medellín, Colombia
Junio 2017

# Índice general

A mi asesor, mi familia, mi novia y amigos, por todo el apoyo que me dieron

# Agradecimientos

Doy gracias Dios por haberme permitido llegar hasta acá y por haberme proporcionado la salud para lograr mis metas. Agradezco especialmente a mi asesor Juan Guillermo Lalinde, por el apoyo, dedicación, paciencia y consejos que me ayudaron a salir adelante en todo momento. A toda mi familia y a mi novia, quienes a pesar de los malos momentos siempre estuvieron ahí para levantarme el ánimo. Muchas gracias a mis amigos, Juan David Pineda, Juan Francisco Cardona y Sergio Monsalve por toda la ayuda que me brindaron. Y a todas aquellas personas que de una u otra forma hicieron que todo esto fuera posible.

*Pensar es el trabajo más difícil que existe. Quizá sea ésta*

*la razón por la que haya pocas personas que lo practiquen*

Henry Ford

# 1

# Introducción

Cada semestre las universidades enfrentan el problema de la programación de horarios y la asignación de aulas de clase de los cursos que se dictan. El problema de asignar horarios de clase en las universidades es un problema complejo, debido a la cantidad de variables y de restricciones que maneja, puesto que el número de combinaciones posibles que pueden generar las variables es muy grande y puede llevar mucho tiempo en encontrar la solución si se quiere

llegar a una solución óptima.

Existen gran cantidad de métodos para encontrar una o más soluciones, pero el obtener la óptima, tiene sus inconvenientes. Como se dijo antes, el número de combinaciones que hay en un problema es muy alto, y para poder encontrar la solución óptima podría llevar mucho tiempo, así que esta primera aproximación para encontrar una programación de los horarios de clase y de las aulas no es viable para ninguna universidad. En estos casos donde es poco factible obtener la solución en un tiempo adecuado, existen métodos que si bien no entregan la mejor solución, buscan encontrar soluciones que cumplan de manera satisfactoria la mayor cantidad de restricciones del problema y obtener estas soluciones en poco tiempo.

Con el avance que se ha tenido en años recientes en la computación, se ha acelerado el desempeño de muchos algoritmos, lo que ha traído consigo el poder explorar la solución de muchos problemas que antes eran imposibles de tratar. Puesto que antes se trataba un problema y su solución con algoritmos secuenciales, ahora muchos problemas pueden ser divididos en pequeños problemas, en donde cada uno de ellos se resuelve en forma paralela, así cuando se unen las soluciones de cada uno de los problemas divididos, da la solución final, pero en menor tiempo.

Hacer una buena planificación para una universidad trae beneficios para la ella, como la no sobrecarga de grupos programados en un horario, balanceo del número de cursos que se le deben de programar a los profesores y el uso eficiente de las aulas. Lo que puede ayudar a

disminuir los picos de congestión tanto en la cafeterías como en los parqueaderos.

Actualmente la Universidad de EAFIT realiza la planificación basándose en la programación de los semestres anteriores. Esto no es bueno y puede llegar a ineficiencias o incluso, en algunos casos, a incumplir con los requerimientos básicos para el buen funcionamiento, ya que el número de matriculados, las materias dictadas y los profesores que dictan las materias, cambian cada semestre, al igual que los horarios de disponibilidad.

Es por este motivo que se desarrolló el presente proyecto de grado, en donde se propone un modelo para la planificación de los profesores, aulas grupos y alumnos, con el fin de proporcionar a la Universidad EAFIT una solución matemática que pueda generar una planificación muy buena, es decir, que entregue la mayor cantidad de grupos, profesores, aulas y alumnos programados en los horarios en los que en la universidad se puede dictar clase y esta solución se entregue en poco tiempo.

La estructura del documento es la siguiente. En el capítulo 2 se habla sobre la Universidad EAFIT dando una breve descripción de ella, como ha sido la programación de horarios como institución educativa y los retos que ha tenido que afrontar para encontrar una solución. En el capítulo 4 se hace un repaso de los antecedentes de la programación de horarios en universidades, y de distintos métodos de solución que se han propuesto a través del tiempo y de su uso en otros campos además del académico. En el capítulo 3 se presentan los conceptos genéricos asociados a la planificación de horarios, qué es la planificación y sus variantes, los

diferentes tipos de solución que hay y se dan algunos ejemplos de su uso. Luego se exponen los algoritmos, mostrando las características de ellos y como es su uso en la resolución de problemas. Por último, se habla sobre la computación paralela y de como esta puede ayudar en el desempeño de los algoritmos y de la mejor solución de algunos problemas. En el capítulo 5 se expone de manera detallada la solución propuesta en el modelo de la programación de horarios, mostrando porqué este es una buena solución para el problema de la programación de horarios y cuáles son los problemas que pueden ocurrir en ciertos casos y su solución. La implementación del modelo propuesto se muestra en el capítulo 6, allí se hablará de las diferentes soluciones desarrolladas y se explicará de manera detallada los resultados obtenidos. En el capítulo 7 se exponen los resultados de la ejecución de las soluciones desarrolladas, y se analizan los valores obtenidos. En los capítulos 8 y 9 se presenta las conclusiones del trabajo y las diferentes propuestas que han surgido como líneas de trabajo futuro de este trabajo.

*Nada es más nocivo para la creatividad que el furor de la inspiración*

Umberto Eco

# 2

# Motivación

## 2.1. Universidad EAFIT

La Universidad EAFIT es una universidad privada cuya sede principal está ubicada en Colombia, en la ciudad de Medellín (Antioquia). Fue creada en 1960, como la "Escuela de Administración y Finanzas" -EAF- por un grupo de empresarios con el propósito de formar profesionales aptos y conscientes de su compromiso en el manejo apropiado de las organiza-

ciones y del crecimiento del país. En 1962 pasa a ser un Instituto Tecnológico, se empieza a llamar "Escuela de Administración, Finanzas e Instituto Tecnológico" - EAFIT, y se comienzan a ofrecer, además de las carreras en Administración y Finanzas, algunas carreras medias técnicas. En 1971 recibe el reconocimiento oficial como Universidad por el Ministerio de Educación Nacional mediante el Decreto 759 del 6 de mayo de 1971 y se empieza a denominar "Escuela de Administración y Finanzas y Tecnologías" - Universidad EAFIT. En 1975, se suspenden las carreras medias para transformarlas en carreras profesionales de ingeniería.

El campus Principal de la Universidad se encuentra ubicado en el sector de El Poblado, y junto con las sedes de Bogotá, Pereira y Samelia (Rionegro - Antioquia) conforman la universidad.

La sede principal de Medellín tiene una extensión de 127.792 metros cuadrados que albergan bloques de aulas, oficinas, laboratorios y otros lugares para la realización de diversas actividades académicas y culturales. También cuenta con camerinos, duchas y baños, además de amplios espacios para el ocio y el descanso, que suman en total 26.467 metros cuadrados. Además, del Centro de Acondicionamiento Físico (CAF) Vivo. Otro de los grandes atractivos de la Universidad son las zonas verdes que proliferan en el campus y que representan 28.215 metros cuadrados. Estas áreas están pobladas de árboles nativos de la región, como pimientos, carboneros y guayacanes.

La Universidad EAFIT cuenta actualmente con un total de 21 pregrados, cerca de 70 espe-

cializaciones, además de 34 maestrías y 6 doctorados; los cuales conforman la oferta académica que ofrecen sus escuelas. La universidad cuenta con 360 docentes de planta, mas de 600 profesores de cátedra, 275 profesores de idiomas y 18 docentes temporales. La población estudiantil esta conformada por unos 11000 alumnos estudiantes de pregrado y unos 3200 de posgrados.

En la actualidad se cuenta en la Universidad EAFIT con la Acreditación de Alta Calidad. La Acreditación es el acto por el cual el Estado hace público el reconocimiento, comprobado por una visita de pares académicos, de la calidad de los programas académicos, la organización, el funcionamiento y el cumplimiento de la función social de una institución de educación superior. La acreditación institucional es, entonces, un reconocimiento de alta calidad que se otorga a las entidades de educación superior que poseen las características de calidad definidas por el Consejo Nacional de Acreditación(CNA). No lo tienen todas las instituciones de educación superior del país, sino únicamente aquellas que voluntariamente se sometan al proceso de evaluación por parte de pares académicos calificados, y que satisfagan exitosamente los criterios exigidos para otorgarla. Desde el año 2003 que comenzó el proceso, y hasta febrero de 2009 sólo 15 universidades colombianas, entre públicas y privadas, habían sido acreditadas como de alta calidad. Según CNA[1], en el 2016 son 38 las universidades acreditadas.

La Universidad EAFIT obtuvo su primera acreditación institucional por un período de 6 años en el año 2003. En 2010, mediante la resolución 1680 del 16 de marzo, se le renovó la acreditación por 8 años más, al considerar los pares evaluadores “que la citada universidad

cumple en alto grado los requerimientos de calidad establecidos por el Consejo Nacional de Acreditación para efectos de la renovación de la Acreditación Institucional".

A esto se suma que todos sus programas de pregrado están acreditados como de alta calidad (que es distinto del registro calificado que sólo evalúa estándares mínimos de calidad). En cuanto a los posgrados, EAFIT cuenta con 5 posgrados acreditados. Finalmente, en 2015 clasificó en el puesto 69 del Ranking QS Latinoamérica, reconocimiento que la deja como una de las 100 mejores universidades de la región.

## 2.2. Asignación de aulas en la Universidad EAFIT

Desde sus inicios, la Universidad EAFIT se preocupó por poder organizar las clases que cursaban sus alumnos de la mejor manera posible en las instalaciones disponibles. Uno de los factores propios de la Universidad EAFIT que caracterizan este proceso es que, a diferencia de muchas instituciones de educación superior, la infraestructura es compartida por todas las escuelas. Esto quiere decir que no se cuentan con aulas dedicadas para cada escuela y que la programación debe realizarse para toda la universidad de manera simultánea. Hasta 1991, el proceso de asignación de aulas era realizado por el personal de la oficina de Admisiones y Registro, quienes durante unos días estudiaban la mejor forma de asignar las aulas de acuerdo al número de alumnos matriculados y a los profesores disponibles.

En 1991, gracias al trabajo de grado de Juan Guillermo Lalinde Pulido y Bertha Lucía Velás-

quez Serrano[46], se diseñó un algoritmo para la asignación de aulas en la Universidad EAFIT que buscaba encontrar la manera de asignar profesores, aulas y grupos, de la mejor manera, haciendo que quedara la mayor cantidad de estos asignada. Dicho proyecto fue dirigido por Luis Eduardo Gómez, quien en esa época era el director de la oficina de Admisiones y Registro y el responsable de elaborar la programación académica, y contó con el apoyo en tecnología de Ana María Barrera, ingeniera del Centro de Informática.

El sistema fue probado por primera vez en el segundo semestre de 1992, programando el 92 % de los grupos. Dicho sistema se utilizó sin modificaciones hasta finales de los 90, cuando la oficina de Admisiones y Registros decidió regresar a la programación manual con el argumento de que la programación se había estabilizado y variaba muy poco de un año a otro. La idea era que se partía de la programación del año inmediatamente anterior y se modificaba para ajustarse a los nuevos cambios.

En el año 1998 se flexibilizan los programas académicos y por primera vez aparecen las materias electivas en las humanidades, las áreas de énfasis profesional y la articulación de los programas de pregrado y posgrado que a la postre conducirían a la formulación del sistema metro vigente en la actualidad. Debido a que el modelo matemático propuesto anteriormente[46] se apoyaba en las restricciones propias de la reglamentación en esa época, y al hecho de que el modelo dejó de utilizarse, no se continuó el trabajo en el mismo, haciendo que el modelo sea obsoleto para las condiciones actuales de la universidad.

Posteriormente, además de estos cambios, se dieron cambios en la intensidad horaria de las clases como resultado de la aplicación del decreto 2566 de 2003[82] y posteriormente del 1295 de 2010[83]. Esto condujo a la alteración del sistemas de franjas horarias en el que se basó el modelo anterior, lo que condujo a una situación en la que había que repensar el modelo desarrollado en[46] para que se pudiera adaptar a las nuevas condiciones de la Universidad EAFIT.

Actualmente el proceso de asignación de aulas se hace con base a la programación de los semestres anteriores, y la única instancia autorizada para modificar la programación es la oficina de Admisiones y Registro. Esta forma de trabajar no sólo no garantiza que se esté haciendo una buena distribución de los espacios de la universidad, sino que es poco flexible a la hora de que los docentes busquen horarios diferentes ofrecidos a los de los semestres anteriores ante la imposibilidad de realizar ajustes de horarios.

Es importante tener presente que el modelo educativo de la Universidad EAFIT trabaja con grupos entre 14 y 40 estudiantes. Si en una materia hay más de 40 estudiantes, se abren tantos grupos como sea necesario de manera que en cada uno de ellos haya más de 14 y menos de 40 estudiantes. Este modelo es diferente de muchas universidades norteamericanas y europeas, en las cuales sólo se programa un grupo por materia y se tienen grupos masivos en los cuales el docente es apoyado por un grupo de asistentes(*Teaching Assistants*), quienes acompañan a los estudiantes en actividades prácticas y les resuelven las dudas.

## 2.3. Trabajos previos en la Universidad EAFIT

La propuesta de[46] fue específica para la Universidad EAFIT y se apoyaba en la políticas que regían la programación de la universidad en la década de los 90. El modelo original tenía en cuenta restricciones propias de la reglamentación de esa época, algunas de las cuales no son válidas actualmente. Las restricciones que siguen siendo válidas son:

- Un profesor no puede dictar dos grupos a la misma hora.
- En un aula sólo se puede dictar una clase a la vez.
- Todo grupo debe ser programado en un aula que se ajuste al tipo de aula requerida y que tenga capacidad para acoger a todos los estudiantes.

La siguiente restricción ya no es válida:

- La universidad garantiza que si un estudiante matricula todas las materias de un semestre perfecto, no va a encontrar ningún conflicto.

Antes se tenía la noción de semestre perfecto, el cual facilitaba el cálculo del modelo pues no era necesario tener presente las materias que registraba cada estudiante. En esa época, cuando se realizaba la programación académica, la asignación del número de grupo a cada grupo que se ofrecía determinaba a qué semestre pertenecía. Ejemplo, el curso de *Cálculo I*, podía tener varios grupos. Los grupos 3x eran del programa de Ingeniería de Sistemas, los 4x de Ingeniería Civil y así sucesivamente. El semestre perfecto se conformaba con todos los grupos de materias del semestre que tuvieran el mismo número de grupo.

A manera de ejemplo, si en Ingeniería de Sistemas había tres grupos de *Cálculo I*, se numeraban 31, 32 y 33. Todos los grupos 31 de las materias del primer semestre, al que pertenecía el curso de *Cálculo I*, conformaban un semestre perfecto. Todos los 32 otro y así sucesivamente. Es importante notar que si no había el mismo número de grupos de todas las materias, lo que ocurría con frecuencia, se podrían tener semestres perfectos de un mismo programa y semestre con diferente número de materias, lo que hacía que el modelo no garantizara que la matrícula siempre se podría llevar a cabo.

Las principales características del modelo desarrollado, que se exponen con mayor detalle en el capítulo 5, son las siguientes:

- Es un modelo aproximado, basado en un algoritmo codicioso(voraz) de orden polinomial.
- La selección del grupo a programar se basa en qué tan difícil es programar el grupo. Los más difíciles se programan primero.
- La selección del horario se basa en identificar cuál de las horas posibles de asignación del grupo tiene menor efecto en los demás grupos que no han sido programados.

En el año 1998, en el proyecto de grado realizado por Lina Vargas y Catalina Pineda se propuso una paralelización del modelo original[67]. Dicho ejercicio se realizó como un trabajo académico pues la dirección de Admisiones y Registro no estaba interesada en implementar nuevamente el modelo. En esa época habían dos tecnologías que competían por ser adoptadas como el estándar para desarrollos distribuidos:

PVM- *Parallel Virtual Machine*: Es un sistema que permite integrar máquinas heterogéneas como si fueran una sola máquina paralela [64]. Un elemento fundamental es la noción de máquina virtual y la compatibilidad entre implementaciones de diferentes proveedores. Para el año 1998 era el estándar de facto en sistemas distribuidos.

MPI - *Message Passing Interface*: Es un estándar diseñado para obtener comunicaciones eficientes en sistemas distribuidos [4]. Su primera versión fue publicada en el año 1992 y para el año 1998 estaba comenzando a tomar fuerza como estándar para el desarrollo de sistemas distribuidos. Se caracteriza por su eficiencia para el manejo de comunicaciones, aunque no hay provisiones para la compatibilidad entre diferentes vendedores.

En [33] se realiza una comparación de ambas propuestas, donde se explica en detalle el funcionamiento de ambas propuestas y se analiza en qué circunstancias es uno mejor que otro.

En [67], se optó por explorar el uso de protocolos de comunicaciones flexibles diseñados para ambientes de alta eficiencia. Por esta razón, la paralelización del modelo se realizó utilizando el protocolo XTP [79]. Dicha implementación se centró en la paralelización y no propuso ninguna mejora al modelo matemático.

*Para toda tesis, existe una antitesis igualmente válida*

Immanuel Kant

# 3

# Marco Teórico

En este capítulo se expone la planificación de horarios en una universidad, cuál es la definición de éste? y el porqué éste es un problema difícil de resolver?. Se explica sobre los diferentes tipos de planificación que pueden surgir en una universidad o institución educativa, las restricciones que se debe de tener en cuenta a la hora de hacer la planificación y como restringe ésta a la solución del problema. Se habla sobre los algoritmos, su clasificación y el cómo estos

pueden ayudar en la solución de problemas como los de planificación de horarios. Por último se habla sobre la programación paralela y como está hace que soluciones formuladas puedan ser ejecutadas más rápido.

## 3.1. Problema

La humanidad, desde siempre, ha buscado mejorar la forma de dar solución a los problemas que surgen en el desarrollo de sus actividades, ya sean estas económicas, administrativas, políticas o militares. Algunos de los problemas que surgen con mayor frecuencia en la vida diaria son los relacionados con la organización de actividades: en estos se busca hacer que los recursos con los que se cuenta, sean aprovechados de la mejor forma posible para lograr un objetivo específico o llevar a cabo una tarea. El como hacer que con el presupuesto que se tiene para comprar víveres, se pueda conseguir la mayor cantidad de productos en el supermercado, salir en la mañana en carro para el trabajo y buscar cuál sería la ruta más corta o la que tome menos tiempo (no necesariamente la más corta) para llegar, o que en un día cualquiera se puedan realizar la mayor cantidad de actividades sin que éstas entren en conflicto son algunos ejemplos a mencionar[49].

### 3.1.1. Programación (*Scheduling*)

Cuando se habla de *Programación*, se hace referencia a un proceso de toma de decisiones donde el objetivo es el de asignar recursos limitados a diferentes tareas que se realizan en el

transcurso del tiempo. *Muller*[38] define la programación como *"La asignación, sujeta a restricciones, de recursos a objetos que están asignados en un espacio y en un tiempo"*. Así que fundamentalmente el *scheduling* tiene que ver con la asignación óptima de recursos a actividades a través del tiempo[5].

Los recursos pueden ser equipos o máquinas en una planta, rutas en un aeropuerto, equipos en una construcción, unidades de procesamiento en un entorno computacional. Las actividades pueden ser operaciones en un proceso de producción, despegues y aterrizajes en un aeropuerto, etapas en un proyecto de construcción, ejecuciones de un programa de computador, entre otros.

Debido a la variedad de los diversos tipos de problemas que se pueden presentar en una situación real, esta se puede clasificar de acuerdo a la caracterización hecha por *Wren*[88]:

Programación (Scheduling): Es la ubicación, sujeta a condiciones de recursos restringidos en el tiempo, para buscar minimizar el costo de uso de estos.

Listado, Catalogación (Rostering): Ubicación de recursos en determinados periodos siguiendo un patrón.

Secuenciación (Sequencing): Encontrar un orden para la ejecución de las actividades.

Planificación (Timetabling): Es la ubicación de recursos restringidos, en donde se busca que se cumpla la mayor cantidad de objetivos dados.

### 3.1.2. Planificar (*Timetabling*)

La planificación o *Timetabling* puede ser definida como el proceso de toma de decisiones para alcanzar un futuro deseado, teniendo en cuenta la situación actual y los factores internos y externos que pueden influir en el logro de los objetivos[26]. Otros hablan de ella como un problema que conlleva trabajos o actividades que deben de ser establecidas en espacios que están restringidos a ciertas condiciones que pueden o no ser cumplidas y que de manera óptima deberá cumplir una función objetivo. Para el presente trabajo se toma la definición dada por *Wren*: "*La planificación es la asignación sujeta a restricciones de los recursos otorgados, con el propósito de ser establecidos en un espacio de tiempo, de tal manera que satisfaga lo más cercanamente posible el conjunto de objetivos deseados*"[88].

El problema de asignación de horarios en una universidad es un ejemplo de este tipo de problemas. En él se busca asignar un horario y un aula a cada grupo, de manera que los profesores puedan dictar todas las materias que se ofrecen en la universidad, esto implica que el grupo quepa en el aula y no esté asignado en ese horario y que el profesor tampoco tenga otra asignación de otro grupo en ese horario.

Todo problema de planificación puede ser dividido en tres partes[43]:

El entorno del problema ¿Cuál es el problema de planificación que va a ser tratado? ¿Cuáles variables o criterios hay que tener presentes?

El criterio de planificación ¿De qué forma se va a buscar la solución del problema? ¿Cuál es el orden que deben seguir los eventos para la construcción de la solución?

Las restricciones del problema ¿Cuáles son las características principales que se deben tener en cuenta cuando se esté buscando la solución del problema? ¿Cuáles son las relaciones entre las variables y que reglas debe de seguir durante el proceso de la búsqueda de la solución?

### 3.1.3. Tipos de Soluciones

Las soluciones a los problemas de la programación y planificación, son clasificados en dos categorías: *las soluciones exactas* (Sec: 3.8.1), las cuales dan una solución óptima, y *las soluciones de aproximación* (Sec: 3.8.2), las cuales entregan una o múltiples soluciones aproximadas [65]. En la sección 3.8 se describe detalladamente.

## 3.2. Planificación de horarios

La *organización* o *planificación del tiempo u horarios* es uno de los problemas más usuales que se ven en la planificación, tanto en el ámbito personal para organizar una agenda de trabajo, como en el empresarial para la organización del despacho de una producción. La planificación de horarios es una herramienta indispensable, ya que sin ella muchas actividades, como organizar las rutinas diarias, no se podrían llevar a cabo. Este tipo de problemas puede

se resuelve en la mayoría de los casos con la ayuda de una agenda ó con de un asistente personal i.e. la secretaria. Sin embargo, para actividades más complejas en las que se busca asignar diversas tareas y recursos, el uso del papel o de una persona podría llevar a que sea imposible de realizar[42]. Algunos ejemplos a mencionar son:

- La programación de los semáforos en una ciudad para que ayuden a manejar el flujo vehicular en una ciudad
- La organización de los vuelos en un aeropuerto
- La programación de la línea de producción en una fábrica
- La programación de las clases en una universidad
- La entrega oportuna de un pedido en una empresa de mensajería

Hacer la programación de las anteriores actividades de forma manual puede hacer que lleve mucho tiempo o incluso a que no sea posible resolverlas en un tiempo razonable, ya que la mayoría de estos problemas tienen muchas variables y restricciones que deben ser tenidas en cuenta. Así que para poder resolverlas se requiere en la mayoría de las veces, de muchas personas y de muchos recursos, y en algunos casos será necesario recurrir a técnicas avanzadas que ayuden a encontrar una solución.

## 3.3. La planificación de horarios en una Universidad

La asignación de horarios en una universidad es un caso especial de la asignación de horarios[90,15] y a su vez es un problema difícil de tratar por dos razones principalmente. La pri-

mera es por ser este un problema de asignación de horarios, lo que lo hace un problema NP-Completo[32,16,74]. La segunda es que la resolución de este problema depende de las necesidades específicas que la universidad tenga: como por ejemplo el maximizar el número de alumnos por grupo, hacer la asignación en el menor tiempo posible, o utilizar la menor cantidad de espacio físico. Por esta razón es poco probable que dos universidades tengan las mismas necesidades a la hora de buscar una solución al problema de la asignación de horarios, haciendo que una solución dada para una institución no sea buena para otra[49]. Usualmente el problema de horarios en universidades considera el siguiente conjunto de elementos: asignación de recursos, asignación de tiempo, restricciones de tiempo entre sesiones, capacidad de las salas y continuidad para las sesiones[68].

En la actualidad aún existen universidades que recurren a implementar la asignación manual[15], lo cual es viable si el conjunto de datos con el que se trabaja es pequeño. Pero cuando la institución tiene un número de cursos y de alumnos también grande, el proceso de asignación se vuelve tedioso y largo, e incluso imposible de programar manualmente[68], porque el tiempo que lleva poder resolverlo sería mucho o porque la cantidad de recursos y/o personal necesario para llevarse a cabo no es suficiente[90].

*Schaerf* diferencia tres grandes problemas globales que existen en la planificación de horarios en las universidades[74,38]:

Generación de horarios escolar (School timetabling): Es la programación semanal de todas

las clases de un colegio, evitando que los profesores se comprometan en dos clases al mismo tiempo. También se le conoce como modelo clase/maestro, en el cual existe una versión simplificada del problema, la cual indica que si no existe ninguna restricción, el problema puede resolverse en tiempo polinomial.

Generación de horarios por curso (Course timetabling): Es cuando en un salón de tamaño fijo se pueden dictar clases a diferentes grupos de estudiantes en horas diferentes. Este problema es el que se da normalmente en las universidades y se busca hacer la programación de manera que haga el uso más eficiente posible de los salones. La principal diferencia con el problema del apartado anterior consiste en que en esta categoría pueden existir estudiantes en común. Si dos cursos tienen estudiantes en común entonces se dice que están en conflicto, y no pueden ser programados en el mismo periodo de tiempo

Generación de horarios por examen (Exam timetabling): Aquí se busca hallar una solución de tal manera que el conjunto de cursos que tienen examen, no se crucen o se crucen lo menos posible con los alumnos que tienen exámenes comunes, esparciendo los exámenes para los estudiantes lo más posible[47,41,30].

*Rhydian Rhys*[49] define el *university course timetabling problem* como la tarea de asignación de eventos, como clases, exámenes o citas, a ser limitados por unos periodos de tiempos

y de tamaños de aulas en concordancia con unas restricciones, definición que es la adoptada en este trabajo de grado.

### 3.3.1. Restricciones

*Corne*[19] ha propuesto 5 categorías de restricciones ante la gran variedad que existen para los problemas de planificación:

Restricciones unarias Estas son las restricciones que implican un solo evento como "las clases no pueden ser programadas antes de las 6 de la mañana", o "se debe programar los cursos en cierto intervalo de tiempo".

Restricciones binarias Son las restricciones que implican dos eventos y un orden en las restricciones, como por ejemplo "Una materia en diferentes grupos que dictar un profesor no la puede ser impartida a la misma hora en grupos diferentes" ó "Un estudiante no puede tomar el curso $A$ antes de cursar el curso $B$".

Restricciones de capacidad Estas son las restricciones que se dan usualmente por tamaño, como que "el número máximo de alumnos que puede contener un salón de clase".

Restricciones de separación de eventos Son aquellas restricciones que requieren que las actividades estén separadas o sigan algún patrón en el tiempo. Algunos ejemplos son las impuestas por políticas de la institución de respetar asignaciones de horarios en patrones predefinidos o las condiciones de no tener horas intermedias vacías.

Restricciones asociadas a los agentes  Estas son las restricciones que se imponen con el fin de promover preferencias, por ejemplo: "a un profesor le gusta enseñar sólo en la mañana", o "al otro profesor le gusta tener cierto número de horas disponibles a la semana".

Cuando todas las restricciones no se pueden satisfacer, es usual hacer otra categorización de ellas, dividiendolas entre: *las restricciones fuertes* y *las restricciones débiles*[11,90,15].

Las restricciones fuertes:  Son restricciones que se deben cumplir obligatoriamente y el hecho de que una de éstas no se cumpla, la solución del problema no tiene ninguna validez. Es por lo tanto, indispensable que se cumplan todas ellas.

Las restricciones débiles:  Son restricciones que es deseable cumplir, pero que podrían ser ignoradas. El hecho de no cumplirlas puede causar inconvenientes en la solución, pero esta sigue siendo aceptable. Puede hacerse una categorización de estas restricciones, para evaluar cuales causarían menos repercusiones en la solución del sistema.

Para resolver el problema particular de una universidad, se deben identificar todas las restricciones y construir la categorización de las restricciones fuertes y restricciones débiles[49].

## 3.4. Algoritmos

Un algoritmo es un conjunto de instrucciones o pasos utilizados para realizar una tarea o resolver un problema. Formalmente, un algoritmo es una secuencia finita de operaciones que

se realizan sin ambigüedades, y que producen en un tiempo finito, una solución a un problema. En pocas palabras, un algoritmo es un procedimiento paso a paso para los cálculos a partir de un estado inicial y una entrada inicial, las instrucciones describen un cómputo que cuando se ejecuta, se procede a través de un número finito de estados sucesivos bien definidos que dan una salida y terminan con un estado final. La transición de un estado a otro no es necesariamente determinista[*]. Algunos algoritmos, conocidos como algoritmos aleatorios, incorporan aleatoriedad en el procesamiento. Los programas computacionales contienen algoritmos que especifican las instrucciones se deben llevar a cabo en un orden específico para realizar una tarea específica. Así pues, un algoritmo puede ser considerado como "*una secuencia de pasos u operaciones finitas para realizar una tarea*".

## 3.5. Tipos de Algoritmos

Existen gran variedad de algoritmos que por sus características y propiedades pueden ser clasificados. En primer lugar, se puede diferenciar entre *algoritmos deterministas* y *algoritmos no deterministas*. Un *algoritmo es determinista* si para unas entrada fija, todas las ejecuciones del algoritmo producen el mismo resultado final. Mientras que un *algoritmo es no determinista* si se introduce algo de aleatoriedad en el proceso de encontrar la solución, por lo que los resultados no necesariamente tienen que coincidir. Existe otro tipo de clasificación basada en

[*]Un algoritmo determinista es un algoritmo que, en términos informales, es completamente predictivo: las entradas determinan la salida.

el tipo de solución que entrega:

Algoritmos exactos: Son algoritmos que siempre entregan una solución óptima.

Algoritmos aproximados: Son los que producen una o varias soluciones que son cercanas a la óptima.

Algoritmos heurísticos: Son algoritmos que entregan soluciones sin ninguna garantía de ser óptimos, pero que generalmente tienen un tiempo de ejecución mucho menor que los aproximados.

Los algoritmos exactos tienen el problema ser muy demorados en su ejecución, incluso cuando el conjunto de datos a procesar es pequeño, por lo que en estos casos se puede usar mejor algoritmos aproximados, pero si la solución aún es demorada, los heurísticos son la mejor opción. La construcción de soluciones es muy diferente entre los algoritmos heurísticos, exactos y aproximados. Los algoritmos heurísticos son algoritmos que imitan fenómenos simples observados en la naturaleza. Estos algoritmos tratan de adaptar comportamientos que son exitosos en la naturaleza para generar soluciones a problemas de optimización complejos. Tratan de alcanzar una solución óptima global, pero no garantizan alcanzarla. Los algoritmos aproximados buscan alcanzar de manera rápida una solución que sea cercana a la solución óptima y los exactos se encargan de buscar y garantizar una solución óptima.

## 3.6. Complejidad Algorítmica

A la hora de trabajar con algoritmos, surge la necesidad de saber qué tan bueno es un algoritmo. Además nos interesa que los algoritmos sean "rápidos" y para saber esto se puede hacer un análisis de ellos. El objetivo del análisis es determinar cómo crece el consumo de los recursos a medida que aumenta el tamaño del problema. Si bien un algoritmo se comporta de manera diferente para cada tipo de recurso –memoria, tiempo o energía–, lo normal es evaluar como crece el tiempo de ejecución a medida que aumenta el tamaño (datos) del problema. Sólo cuando hay problemas severos de almacenamiento se analiza el consumo de memoria. Finalmente, el análisis de consumo de energía es útil para el trabajo con dispositivos móviles, donde es importante ver el desgaste de energía frente al uso. Sin embargo hay que resaltar que cuando se trabaja con modelos matemáticos lo más recomendable es utilizar *clusters* de computadores para estudiar cual es su rendimiento.

La clasificación de los algoritmos se puede hacer de diferentes maneras. Una primera de ellas es hacerlo a partir de la forma en como se evalua un algoritmo:

Empírico: Consiste en estimar el comportamiento del algoritmo en la practica, probándolo con varios ejemplos del problema y graficando cómo cambia el tiempo de ejecución cuando cambia el tamaño del problema.

*Ventaja:* No requiere formalizar el algoritmo matemáticamente ni es necesario do-

minar las técnicas de solución de recurrencias, i.e. ecuaciones de diferencias.

*Inconvenientes:* El desarrollo del algoritmo depende del lenguaje de programación, el compilador, equipo y la habilidad del programador. Llevar a cabo un análisis empírico serio por lo general conlleva mucho tiempo. Es muy difícil comparar los algoritmos a través de pruebas empíricas, ya que el rendimiento puede depender de las instancias del problema elegidas para las pruebas.

Analítico: Consiste en determinar de manera formal cómo crece el consumo de recursos a medida que aumenta el tamaño del problema. Con el fin de permitir una comparación entre los diferentes algoritmos, se asume que el tiempo de ejecución de una instrucción simple es constante y se analiza el comportamiento asintótico cuando el tamaño del problema tiende a infinito. Está basado en la programación estructurada, la cual es una demostración de que todo algoritmo puede ser reescrito de manera que sólo utilice ciclos, secuencias y decisiones[24,59].

*Ventaja:* Los resultados obtenidos permiten comparar de manera eficiente los algoritmos y, adicionalmente, sirve para determinar que tanto escala un algoritmo, es decir, hasta cuanto puede crecer el tamaño del problema y tener respuesta en tiempo razonable.

*Inconvenientes:* El cálculo analítico puede ser muy complicado.

Otra manera de clasificar los algoritmos se puede hacer desde el enfoque analítico, en donde se presentan tres casos que dependen de los condicionales y ciclos utilizados, de cual es el consumo de recursos de este, y de cual es el tiempo de ejecución con respecto a el número de datos de entrada:

Caso promedio: Consiste en estimar el número medio de pasos que necesita el algoritmo. Para poderlo realizar de manera adecuada, se debe conocer la distribución de probabilidad de los datos de entrada, los posibles problemas y las técnicas estadísticas que se utilizaran para calcular el tiempo de ejecución del algoritmo.

*Ventaja:* Permite tener la mejor estimación posible de la forma como crece el consumo de recursos a medida que crece el tamaño del problema.

*Inconvenientes:* El análisis depende fundamentalmente de la elección de la distribución de probabilidad. A menudo es difícil determinar la distribución de probabilidad de los datos para los problemas prácticos. El análisis suele ser bastante complejo matemáticamente, lo que hace muy difícil llevar a cabo problemas complicados.

Peor caso: Consiste en identificar el límite superior para el número de operaciones que va a requerir el algoritmo para cualquier entrada. En este análisis siempre se asume que, en los condicionales, el código que se ejecuta es el de mayor complejidad.

*Ventaja:* Por la forma como se calcula, determina una cota superior para el consu-

mo de recursos.

*Inconvenientes:* Podría llegar a clasificar como malo un algoritmo que solo funciona mal en casos muy específicos.

Mejor caso: Consiste en identificar el límite inferior para el número de operaciones que va a requerir el algoritmo para cualquier entrada. En este análisis siempre se asume que, en los condicionales, el código que se ejecuta es el de menor complejidad.

*Ventaja:* Por la forma como se calcula, determina una cota mínima para el consumo de recursos y ayuda a identificar las condiciones mínimas para que el algoritmo pueda ser utilizado.

*Inconvenientes:* Al ser una cota mínima, no da información sobre el uso real de recursos y puede llegar a clasificar un algoritmo como bueno, cuando solo funciona muy bien en casos muy específicos.

## 3.7. Complejidad Computacional

Si bien el análisis de la complejidad de los algoritmos es importante, este está orientado a la complejidad de la solución y no a la complejidad propia del problema. El estudio de la complejidad propia del problema se le denomina "complejidad computacional". A continuación se define algunas de las clases de complejidad computacional, las cuales se utilizan para clasificar los problemas de acuerdo con su "dificultad"[49]:

Clase P: Dado un problema, decimos que pertenece a la clase de complejidad P si existe al menos un algoritmo que resuelve de manera óptima cualquier instancia del problema en tiempo polinomial.

Clase NP: Dado un problema, un algoritmo pertenece a la clase de complejidad NP si existe un algoritmo no determinista que encuentra una solución a problemas en tiempo polinomial. La principal característica de los problemas NP es que los algoritmos conocidos que solucionan de manera exacta el problema son exponenciales o factoriales, pero la verificación de una solución cuando esta es válida se puede hacer en tiempo polinomial.

Clase NP-completo: Un problema es NP-completo si cualquier problema en clase NP se puede reducir a él en tiempo polinomial. Esta clase de problemas es muy importante porque de encontrarse un algoritmo polinomial para un problema NP-completo se demostraría que NP = P.

Clase NP-duro: Si la versión de decisión de un problema de optimización combinatorial es NP-completo, se dice que la versión de optimización del problema es un problema NP-duro. En un problema de decisión se pregunta por la existencia de una solución con una característica específica. En un problema de optimización se pregunta por la mejor solución posible. Es en este grupo donde se encuentran los problemas de asignación de

horarios [68,13]

La diferencia entre las clases NP-completo y NP-duro es que un problema puede ser NP-duro, sin pertenecer a la clase NP. La gran pregunta de la teoría de complejidad computacional es si las dos clases P y NP coinciden, es decir, si P = NP. Aunque existe un amplio consenso de que ambas clases deben ser diferentes, nadie lo ha probado hasta ahora. En el caso de que ambas clases fuesen iguales, esto tendría un gran impacto en muchos campos, como por ejemplo la existencia de un algoritmo polinomial para factorizar números primos, lo que podría comprometer seriamente muchos protocolos de seguridad.

## 3.8. Tipos de Soluciones Algorítmicas

*Pitol* [68] describe varios tipos de soluciones algorítmicas, pero estas soluciones se agrupan en dos categorías por sus características y su forma de solución: las *soluciones exactas* y las *soluciones aproximadas*.

### 3.8.1. Soluciones Exactas

Son soluciones óptimas. En las soluciones exactas, generalmente la complejidad del algoritmo es exponencial, lo que indica que el tiempo requerido para encontrar la solución crece de manera exponencial con respecto al tamaño de la entrada. Es importante tener presente que los algoritmos de complejidad exponencial sólo son útiles cuando se tienen conjuntos de

datos pequeños. La forma como crece el tiempo de ejecución cuando aumenta el tamaño del problema, implica que para conjuntos de datos moderadamente grandes, el tiempo de ejecución no es razonable y no se puedan utilizar en la práctica para encontrar una solución. Los métodos exactos construyen una solución óptima a partir de los datos del problema, siguiendo una serie de reglas que determinan de manera exacta el orden de procesamiento [65]. Algunas de los métodos propuestos para encontrar soluciones exactas son:

Programación lineal: Es una técnica de investigación de operaciones para la determinación de la asignación óptima de recursos escasos cuando la función objetivo y las restricciones son lineales.

Ramificación y poda (*Branch & Bound*): Esta solución hace uso de una estructura de árbol construida dinámicamente, como forma de representar el espacio de todas las secuencias posibles. La búsqueda comienza en el nodo raíz y continúa hasta llegar a un nodo hoja. Cada nodo en un nivel $p$ de la búsqueda puede representar una secuencia parcial de $p$ operaciones. Desde un nodo no seleccionado, la operación de ramificación determina el siguiente conjunto de posibles nodos a partir del cual puede progresar la búsqueda [65]

Búsqueda exhaustiva: La búsqueda exhaustiva consiste en evaluar cada una de las soluciones del espacio de búsqueda hasta encontrar la mejor solución global. Esto significa que si no se conoces el valor correspondiente a la mejor solución global, no hay forma de

asegurarse de que se ha encontrado la mejor solución utilizando esta técnica, a menos que examines todas las posibles soluciones. Como el tamaño del espacio de búsqueda de problemas reales es usualmente grande, es probable que se requiera mucho tiempo de computación para probar cada una de las soluciones[84]

Backtracking El método de *backtracking* o *retroceso* es una técnica que trabaja buscando continuamente el extender una solución parcial, hasta encontrar una o más soluciones, demostrar que no existe solución, o agotar los recursos de computo. En cada etapa de la búsqueda, si una extensión de la solución parcial actual no es posible, se va hacia atrás para encontrar una solución parcial y trata nuevamente[38,37,84]

Este tipo de soluciones normalmente se usan en la planificación de horarios de escuelas y de colegios en donde el conjunto a programar es pequeño, y las variables a programar son solo los profesores y las aulas[35,36].

### 3.8.2. Soluciones Aproximadas

Son soluciones que no son óptimas, pero pueden encontrar la o las soluciones en un tiempo razonable. Por esta razón hace que este tipo de soluciones sean las más adecuados para problemas de gran magnitud[65]. Algunas de las técnicas más usadas para obtener estas soluciones son los algoritmos heurísticos:

Métodos de búsqueda local (*Local search methods*): Procedimiento basado en la suposición

de que es posible encontrar una secuencia de soluciones entre la solución inicial y la final tal que cada una de ellas es ligeramente diferente a la inmediatamente anterior. Tiene la ventaja de que en poco tiempo puede encontrar soluciones suficientemente buenas para un conjunto amplio de problemas. Este procedimiento puede ofrecer una medida de bondad de la solución encontrada, pero no garantiza que la solución obtenida sea el óptimo global del problema considerado [65,84]

Búsqueda tabú (*Tabu Search*): Este método es un tipo de búsqueda por espacios de búsqueda(entornos), que permite moverse a una solución de entorno aunque no sea tan buena como la actual, de este modo se puede escapar de óptimos locales y continuar la búsqueda de soluciones aún mejores. La forma de evitar viejos óptimos locales es clasificando un determinado número de los más recientes movimientos como "movimientos tabú", los cuales no son posibles repetir durante un determinado horizonte temporal. Así, el escape de los óptimos locales se produce de forma sistemática y no aleatoria [62,6]

Algoritmos genéticos (*Genetic algorithms*): Los Algoritmos Genéticos son métodos adaptativos, generalmente usados en problemas de búsqueda y optimización de parámetros, basados en la reproducción sexual y en el principio de supervivencia del más apto [57,69]

Algoritmos meméticos (*Memetics algorithms*) Son un conjunto de algoritmos que constituyen un paradigma de optimización basado en la explotación sistemática de conocimien-

to acerca del problema que se desea resolver y de la combinación de ideas tomadas de diferentes métdos metaheurísticos[51]

Algoritmos Evolutivos (*Evolutionary Algorithms*): Son un esquema de representación que aplica una técnica de búsqueda de soluciones enfocada a problemas de optimización, inspirada en la teoría de la evolución de Charles Darwin. Se basa en el algoritmo de selección propio de la naturaleza, con la esperanza de que consiga éxitos similares, en relación a la capacidad de adaptación a un amplio número de ambientes diferentes[57]

Colonias de hormigas: Los algoritmos de colonias de hormigas están basados en el comportamiento de las hormigas. Los biólogos y los entomólogos han descubierto la facilidad que tienen las hormigas para encontrar siempre el camino más corto entre el hormiguero y la fuente de alimento, este comportamiento ha sido estudiado y se ha encontrado que las hormigas mantienen una comunicación indirecta con una sustancia volátil llamada feromona. Con la feromona las hormigas son capaces de crear una ruta y a través del tiempo optimizar sus recorridos obteniendo así un camino corto sin una visión global del terreno. Así que estos algoritmos se caracterizan por simular el comportamiento de las hormigas cuando forman las rutas entre el nido y la fuente de alimento, en base a un rastro de feromonas que depositan en la trayectoria efectuada[68,78]

Algoritmos Voraces Los algoritmos voraces son algoritmos en los que se toman decisiones

locales para llegar a una solución óptima parcial. Son también denominado *golosos-miopes*(del inglés greedy-myopic) por las siguientes razones[18]:

* Es goloso o voraz porque siempre escoge como candidato para formar parte de la solución a aquel que tenga mejor valor de la función objetivo con la información disponible en ese momento. Este criterio *goloso* de selección se basa en optimizaciones locales. Cuando los valores óptimos localmente son válidos globalmente, producen resultados exactos.

* Es miope porque esta elección es única e inmodificable dado que no analiza más allá los efectos de haber seleccionado un elemento como parte de la solución. No deshacen una selección ya realizada; una vez incorporado un elemento a la solución, este permanece hasta el final y cada vez que un candidato es rechazado, lo es permanentemente.

Recocido simulado (*Simulated Annealing*): Es una técnica que hace uso de conceptos originalmente descritos por la mecánica estadística. Tiene su base en el proceso físico de recocido, el cual primero reblandece un sólido mediante su calentamiento a una temperatura elevada, y luego va enfriando lentamente hasta que las partículas se van posicionando por sí mismas en el "*estado fundamental*" del sólido. El recocido simulado es capaz de encontrar la solución óptima, sin embargo, esta se alcanzará tras un número infinito de pasos, en el peor de los casos[57]. Por dicha razón, se puede clasificar también

como un algoritmo aproximado

Los algoritmos mencionados en este apartado, son los que se utilizan usualmente en la búsqueda de soluciones para la planificación de horarios en las universidades. Se pueden encontrar otra lista de algoritmos en [6,7,9,17,20].

## 3.9. Computación de Alto Rendimiento

La *computación de alto rendimiento* se refiere al uso de computadoras para resolver problemas que requieren gran capacidad de procesamiento de información. Generalmente involucra el usar un conjunto de recursos de computación en paralelo, llamado *cluster*, que cooperan para solucionar un problema en el cual intervienen gran cantidad de datos y cálculos[12]. Esta área se encuentra vinculada con la investigación científica, pero también con el desarrollo de productos innovadores[56].

Un cluster de computadores es una máquina de procesamiento paralelo, que utiliza hardware débilmente acoplado, es decir, un conjunto de máquinas, donde cada una de ellas se denomina nodo, las cuales se encuentran conectadas entre sí mediante una red de comunicaciones de alta velocidad, las que son coordinadas mediante un equipo especializado llamado nodo maestro o *frontend-node*, en inglés[52]. Usualmente, para hacer uso de la computación de alto rendimiento, se trabaja descomposición de problemas, haciendo que un problema se convierta en muchos problemas más pequeños, de tal modo que cada nodo resuelva un pro-

blema, y al final, la solución total sea construida a partir de las soluciones obtenidas por cada uno de ellos.

## 3.10. Plataformas de Computación Paralela

Los sistemas de computación paralela se clasifican de varias maneras, atendiendo a diferentes especificaciones. Flynn[31] se basa en la posibilidad de procesar uno o mas flujos de instrucciones así:

SISD (*Simple Instruction Simple Data*) Un sólo flujo de instrucciones opera sobre un sólo flujo de datos. Este es el modelo Von Neumann.

SIMD (*Simple Instruction Multiple Data*) Un sólo flujo de instrucciones opera sobre múltiples flujos de datos. Todos los procesadores ejecutan la misma instrucción aunque sobre distintos datos.

MIMD (*Multiple Instruction Multiple Data*) Múltiples flujos de instrucciones operan sobre múltiples flujos de datos. Cada procesador ejecuta su propio código sobre sus propios datos.

MISD (*Multiple Instruction Simple Data*) Múltiples flujos de instrucciones operan sobre un sólo flujo de datos. Es teóricamente equivalente al tipo *SISD*.

Otra clasificación importante que se hace debido a la distribución de la memoria y la forma en que los procesadores acceden a ella son:

Memoria compartida: Todos los procesadores tienen acceso independiente a una memoria común, cada uno de ellos posee una pequeña memoria local para almacenar código y resultados intermedios. La ventaja más importante que hay, es que la comúnicación que hay entre los procesos es muy rápida, por el hecho de compartir la memoria. Tiene el inconveniente de que puede llegar a tener conflictos en el acceso de los datos, así como llegar a un bloqueo permanente por la espera de un datos entre procesadores que nunca llegue.

Memoria distribuida: En estos sistemas cada procesador posee su propia memoria loca inaccesible a los demás. Los procesadores están conectados entre sí, para efectos de intercambio de datos, mediante una red de interconexión. La comunicación entre los procesadores se hace a través de envío de mensajes, lo que lleva a que la ejecución sea un poco mas lenta que la de *la memoria compartida*, pero más segura.

Como se dijo en la sección 2.3, *MPI* es un estándar de programación en paralelo mediante paso de mensajes en sistemas de *memoria distribuida* bajo el modelo *MIMD* que permite crear programas portables y eficientes. Este será usado para hacer la parelización de datos en la programación de horarios en la Universidad EAFIT.

## 3.ii. Aproximación Utilizada

Para este trabajo de grado se trabaja en el *university course timetabling problem* y, cómo el problema de asignación es NP-completo, se construye una heurística para buscar una solución aproximada al problema de asignación de horarios teniendo en cuenta las características propias de la Universidad EAFIT presentadas en la sección 5.1. Hay muchas maneras de buscar una solución al problema de la planificación, pero la escogida en el presente trabajo es construir un modelo matemático que se implementa mediante un algoritmo goloso-miope que se puede paralelizar.

*El hombre no sabe de lo que es capaz hasta que lo intenta*

Charles Dickens

# 4

# Estado del Arte

En el presente capítulo se habla sobre la planificación de horarios, explicando de dónde surge ésta, cómo fue evolucionando a medida que surgían nuevas problemáticas a las soluciones propuestas y sus inconvenientes en diferentes ambientes usados. Se habla también de las soluciones más usadas actualmente en la planificación de horarios y los campos de acción en los que estás soluciones son utilizadas.

## 4.1. Inicios

El estudio del problema de la *planificación* es relativamente nuevo y tiene un historial significativo. La planificación fue usada inicialmente durante la segunda guerra mundial para organizar la entrega de provisiones, víveres y armamento a los ejércitos en las operaciones militares. Esta campaña fue dirigida por *George B. Dantzig* quien no sólo uso por primera vez la *programación lineal*, sino que desarrolló uno de los métodos más importantes en la investigación de operaciones: *el método simplex*. Sus trabajos no fueron publicados sino hasta 1949 por razones de seguridad nacional[54].

Finalizando la segunda guerra mundial, muchos científicos estadounidenses estaban trabajando en investigación de operaciones para el análisis exacto del combate, incluyendo temas como el hundimiento de barcos y submarinos, la evasión de aeroplanos enemigos y el cálculo del momento en el que deberían estallar las bombas en el agua o en el aire[84].

En la década del cincuenta, varios científicos, entre ellos J. R. Jackson, S. M. Johnson y J. Friedman trabajaron en el problema del *Job Shop Scheduling*, que es un tipo de problema de planificación de tareas, en donde uno de los objetivos principales es la disminución de tiempos en la tarea de planificación. Desarrollaron métodos de solución a este problema, como algoritmos de reglas de despacho de prioridad y el de aplicar un modelo de álgebra booleana para representar secuencias de procesamiento[65].

## 4.2. Planificación de Horarios

Pero los primeros estudios formales sobre *planificación de horarios* ó el problema de la *asignación de horarios* fueron dados en 1959 con el trabajo de *Blakesley*[8] donde se utilizó para el registro de estudiantes[21] en la *United State Naval Postgraduate School*. A partir de ese momento se usaron técnicas como la búsqueda n-aria, el coloreado de grafos, la programación lineal y la reducción de matrices[50] para encontrar la solución a este problema. Pero existía en ese momento un inconveniente a estas soluciones, pues estas solo eran utilizadas en centros de estudios superiores o colegios[86,48] en donde la cantidad de datos a evaluar era pequeña[53].

En la búsqueda de soluciones más eficaces para resolver la planificación de horario, a finales de los años sesenta surgieron las técnicas no exactas[43], en donde *Rechenberg* introduce *las estrategias evolutivas* para problemas de optimización. Estas técnicas surgieron gracias a los trabajos de varios científicos que venían trabajando desde la década anterior, y que vieron en la evolución una herramienta que podían emplear para solucionar problemas de optimización en ingeniería. *John Holland*, en 1970 en la Universidad de Michigan, creo un método heurístico de búsqueda y optimización: los *algoritmos genéticos*. Su estudio se centraba en la creación de un algoritmo, mas que en buscar una solución a un problema, específicamente en la forma como se podía adaptar el algoritmo para encontrar la solución. De su trabajo salieron diferentes tipos de algoritmos genéticos como los basados en cruce, en mutación y en inversión[58,28].

En 1975 *Even* y *Itai* [27,32] y *Ullman* [81] demuestran que el problema de la *asignación de horarios* es un problema NP-Completo (problemas que requiere de un tiempo muy elevado para encontrar la solución óptima, incluso en problemas de pequeño tamaño en sus datos [32]) por el número de variables, el tipo de restricciones que maneja, la limitación de tiempo para obtener un resultado y la escasa cantidad de recursos computacionales que se tiene para resolverlo [13,68]. Ante la imposibilidad de encontrar una solución en poco tiempo y de encontrar una solución óptima, se fue motivando el uso de nuevas estrategias de solución, así como el desarrollo de métodos de solución no exactos, heurísticos y metaheurísticos, los cuales son mucho más rápidos en encontrar una solución, con resultado no óptimos, pero si muy cercanos a éste.

Muchas de las soluciones propuestas, como la de DeWerra [22], fueron formulaciones teóricas por la falta de recursos computaciones que había en la época. Pero en los años siguientes el aumento de problemas con grandes volúmenes de datos y con el auge de las computadoras personales, el desarrollo y el progreso de la investigación de algoritmos y de los métodos matemáticos, llevó a que se concentrara gran parte de la investigación del problema en la búsqueda de nuevas soluciones. Algunas de las técnicas que se encontraron y fueron utilizadas fueron: el métodos de Lagrange [80], los modelos de solución matemáticos [55], las bases de datos deductivas [60], el método incremental [87], los sistemas de soporte de decisión [25], la búsqueda tabú [39] y los algoritmos genéticos [29], entre otras.

## 4.3. Planificación de horarios en las Universidades

El problema de la *planificación de horarios en las universidades* es un problema de *asignación de horarios* que presenta complicaciones adicionales puesto que, además de ser un problema NP-Completo por ser un problema de asignación de horarios, en el momento de planificar los horarios, cada universidad tiene sus propias necesidades para buscar una solución. Las necesidades de cada universidad pueden requerir optimizar los horarios de los profesores o de los alumnos, buscar un uso equitativo de las aulas, minimizar el recorrido que deben realizar los estudiantes y profesores entre aulas, o cualquier otra característica que considere necesaria para llevar a cabo la planificación de las clases de la universidad. Por lo tanto, la solución en la mayoría de los casos, es válida para una universidad en especifico. Esto ha hecho que este problema tenga mucha importancia[72] en la comunidad científica y académica[75], y en universidades y centros de investigación alrededor de todo el mundo[76]. Muestra de ello es la creación de certámenes internacionales, en donde se proponen un caso particular con restricciones difíciles de abordar para los participantes. Ejemplos de este tipo de eventos se tiene el "International Timetabling Competition" de *Metaheuristics Networks*[2] y el *PATAT* "International Timetabling Competition"[3].

## 4.4. Soluciones Actuales

En los años recientes la búsqueda de soluciones al problema de planificación sigue activo. Cada año se proponen nuevos métodos para la solución del problema, como son los algoritmos meméticos[10], los algoritmo de búsqueda armónica[44], los algoritmos genéricos híbridos de abejas[66] y las aproximaciones hiperheurísticas[23], entre otro tipo de soluciones[15].

Gracias a los avances actuales en la computación paralela, ha habido una corriente grande entre los científicos que busca soluciones para el problema de la planificación utilizando aproximaciones paralelas, tales como la programación metaheurística paralela[77], la hiperheurística paralela[70] y la adaptación de algoritmos clásicos buscando mejorar su eficiencia con el paralelismo, ejemplo de ello la programación entera[45], los algoritmos de colonias de hormigas[63] y los algoritmos genéticos[89].

### 4.4.1. Otros Campos de Acción

Los resultados obtenidos en la investigación para resolver el problema de la planificación de horarios en las universidades, también tienen a mostrado tener aplicaciones en otros campos[76]. Algunas de las áreas donde más uso han tenido son:

Transporte: Asignación de vuelos aéreos[71] y de rutas de buses[14,40]

Centros educativos: Asignación de clases, profesores y exámenes[68].

Medicina: Distribución de médicos y enfermeras en la atención de pacientes[61,73] y en la programación de cirugías[85].

Industria mecánica: Planificación del mantenimiento de máquinas[28].

Industria del entretenimiento: Planificación del rodaje de películas[34].

*Las matemáticas consisten en demostrar las cosas más obvias de la forma menos obvia*

George Polye

# 5

# Algoritmo para la planificación de aulas en la Universidad EAFIT

En este capítulo se analiza de manera detallada la propuesta para encontrar una solución aproximada al problema de la planificación de horarios. Primero se presenta la forma como se soluciona actualmente en la Universidad EAFIT, y las condiciones y restricciones que tiene

que cumplirse para que sea buena la solución. Luego se detalla cuales son las variables a tener en cuenta y la forma en que el algoritmo va generando la solución. Por último, y debido a que la solución del problema es aproximada, se analizan los posibles casos donde la solución entregada no es la más óptima.

## 5.1. Planificación de aulas en la Universidad EAFIT

El problema de la asignación de horarios en la Universidad EAFIT es una variante de la programación de cursos (*course timetabling*) pues incluye algunas restricciones adicionales. A diferencia de muchas universidades, los cursos en la Universidad EAFIT pueden dictarse en cualquier salón mientras cumpla los requisitos necesarios: que el salón tenga la capacidad requerida y sea del tipo de aula adecuado para ser dictada (laboratorio, sala de computación, aula normal, etc.). La unidad de trabajo son los grupos pues de un mismo curso se ofrecen múltiples grupos.

El modelo económico de la Universidad EAFIT exige que los grupos normales deben tener una capacidad mínima de 14 estudiantes y máxima de 40. El mínimo es para garantizar que hay el punto de equilibrio y que el curso no se dicte a pérdida. El máximo se deriva del modelo pedagógico propio de la universidad. Pese a esto, la universidad se reserva el derecho de programar cursos en los cuales el número mínimo y máximo de estudiantes puede variar. Administrativamente, estos deben ser aprobados por la vicerrectoría.

Para determinar el número de grupos, al finalizar un semestre los estudiantes registran las materias que van a matricular el siguiente semestre y, con base en dicha información, la oficina de Admisiones y Registro determina cuántos grupos de cada curso se van a programar.

La planificación académica, es decir la asignación del horario a cada grupo, debe cumplir las siguientes condiciones:

- En un aula sólo puede haber un grupo a la vez
- Un profesor sólo puede dictar un grupo a la vez
- Los grupos deben ser programados de tal manera que la gran mayoría de los estudiantes puedan tomar todas las materias registradas.
- Un grupo sólo se puede programar en un aula del tipo adecuado y cuya capacidad sea, como mínimo, el número de estudiantes que requiere el grupo

### 5.1.1. Pregunta de investigación

El problema a resolver en el presente trabajo puede ser formulado de la siguiente forma: "De que manera debe ser asignado el horario y el aula a cada grupo de tal manera que ":

- El profesor del grupo no dicte clase en otro grupo a esa misma hora
- En el aula no haya más grupos programados a esa hora
- Existan suficientes estudiantes que puedan tomar la materia en ese horario
- El uso de las aulas sea lo más homogéneo posible
- La capacidad del aula permita acoger al grupo

- El aula sea del tipo requerido por el grupo

El objetivo del problema es programar la mayor cantidad de grupos, optimizando el uso de los recursos de la universidad y satisfaciendo las restricciones y políticas de la misma.

### 5.1.2. Problema de la programación de cursos en la Universidad EAFIT

El modelo propuesto para encontrar una solución aproximada al problema, se basa en la utilización de un algoritmo (voraz)(ver 3.8.2). Se decidió trabajar con un algoritmo voraz porque de esa manera se garantiza que el tiempo de ejecución es polinomial. El reto en el modelo es seleccionar un orden de asignación de manera que la solución sea razonable.

En el caso de la Universidad EAFIT, de una misma materia se pueden ofrecer múltiples grupos. Por tal motivo, la planificación no se realiza a nivel de materia sino a nivel de grupos.

Los estudiantes al final de semestre, hacen una inscripción de las materias que desean cursar en el siguiente. Esta planificación del siguiente semestre se hace teniendo en cuenta el número de estudiantes que inscribieron la materia y los cupos máximos y mínimos definidos por la oficina de Admisiones y Registro. La Universidad trata de garantizar a cada estudiante que pueda encontrar un horario en el cual matricular todas las materias a las que se registró.

Una vez concluido el registro de los estudiantes, se procede a buscar los profesores necesarios para dictar las materias. Cada profesor tiene unas materias para dictar y un horario de disponibilidad para dictarlas. El horario de disponibilidad podría ser afectado, ya sea por la

diferenciación natural entre profesores de planta y profesores de tiempo parcial, o por restricciones propias originadas en actividades diferentes a la docencia que realice el profesor.

Con los datos anteriores se procede a buscar las aulas donde se puede asignar los alumnos que van a ver la materia asignada al grupo, teniendo en cuenta que el aula tenga disponibilidad, sea del tipo adecuado y tenga la capacidad para dictar la materia. Para poder realizar correctamente la asignación, se debe de tener en cuenta las siguientes restricciones.

### 5.1.3. Restricciones del problema

Restricciones duras:

- Un profesor no puede dictar más de un grupo en el mismo horario
- En un aula no pueden ser asignados dos o más grupos en el mismo horario
- Un grupo solo puede ser programado en un horario
- En un aula no se pueden asignar mas alumnos que la capacidad que tenga esta
- Una materia (curso) tiene un número de horas a la semana para ser dictada
- Los grupos sólo pueden ser programados en horarios en los cuales la universidad tenga servicio

Restricciones blandas:

- Los horarios de las sesiones de clase no deben exceder las dos horas
- Los horarios de los grupos deben ajustarse, en lo posible, a la estructura de franjas que maneja la Universidad EAFIT. La política es que las sesiones de clase queden distribuidas a lo largo de la semana.

Esta es la estrategia general del algoritmo implementado: se selecciona el grupo con más restricciones y se programa en el aula con menor demanda. De esta manera, se trata de balancear el uso de los recursos de la universidad al asignar primero los grupos más restringidos en los horarios en el que afecte menos a los que aún no han sido programados.

Se programan inicialmente aquellos grupos que tengan menos opción de escoger horario y dejando para después aquellos que tengan más opciones de ser programados. En cuanto a las aulas, si se asignaran primero las aulas más demandadas harían que grupos que tienen otras alternativas quedaran asignados a ellas haciendo que algunos grupos probablemente no puedan programarse por no tener aulas disponibles, ya que tienen pocas opciones. Al programar primero las menos demandadas, se balancea la demanda sobre las aulas al tratar de ubicar un grupo en el aula donde menos grupos se vean afectados por la planificación.

A continuación se presenta una descripción del algoritmo, luego se muestra el algoritmo implementado y posteriormente se hace el análisis de los casos en los cuales la decisión que toma puede no ser óptima, ya que este utilizar un algoritmo voraz. Debido a la existencia de estos casos donde la solución no es óptima, el algoritmo no es exacto, pero al ser aplicado a datos reales, da una aproximación buena y está dentro de los parámetros esperados.

## 5.2. Algoritmo

La planificación de los grupos en la universidad consiste en asignar a cada grupo, un profesor y sus alumnos, un aula y un horario, de tal manera que tanto el profesor, como el aula y los alumnos tengan disponibilidad para que se pueda dictar una materia en ese horario. El grupo a planificar no puede ser asignado más de una vez, y en este no se puede asignar sino un profesor y un aula, y el horario en el que se programe debe de ser válido para la universidad. El aula a ser asignada debe ser del tipo de aula adecuado y debe tener la capacidad suficiente para acomodar a los estudiantes que se van a matricular. Un estudiante no puede matricularse en más de un grupo en el mismo horario.

La planificación a generarse, debe programar la mayor cantidad de grupos posibles para la universidad, optimizando los recursos disponibles (las aulas y los horarios) y asegurando que la planificación sea realizada en un tiempo de computo razonable.

Sin pérdida de generalidad, se puede utilizar un vector $h$ para representar un horario. Si bien lo convencional es representar el horario como una matriz con días como columnas y horas como filas, para la formulación del problema esta representación no tiene ninguna ventaja y puede disminuir la eficiencia computacional, así que se representa como un vector $h[i]$, tal que $i$ es el intervalo $i$ de tiempo(hora) del horario de la semana.

Se define $h[i]$ como la disponibilidad del recursos que tiene asociado el horario $h$ en la

hora $i$ de la siguiente manera:

$$h[i] = \begin{cases} 0 & \text{está ocupado} \\ \\ 1 & \text{está libre} \end{cases} \tag{5.1}$$

Sean $G = \{g_1, g_2, \cdots, g_{n-1}, g_n\}$ el conjunto de todos los grupos que se van a ofrecer, $P = \{p_1, p_2, \cdots, p_{j-1}, p_j\}$ el conjunto de los profesores, $\mathcal{M} = \{\mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_{l-1}, \mathcal{M}_k\}$ el conjunto de materias y $\mathcal{A} = \{a_1, a_2, \cdots, a_{l-1}, a_l\}$ el conjunto de las aulas.

Cada grupo $g_i$ tiene los siguientes atributos:

- Tipo de aula $t_i$: El tipo de aula en la que debe ser programado
- Cupo máximo $C_i$: Número de estudiantes que va a tener el grupo. Al seleccionar el aula, ésta debe cumplir no sólo con el tipo de aula, sino también que debe tener la capacidad para alojar este número de estudiantes como máximo
- El profesor $p_i$: Es el profesor que va a dictar el grupo. El horario en que se programe el grupo debe ser compatible con el horario del profesor
- Horario $h_{g_i}$: Es una estructura que representa a que horas puede ser programado el curso y se construye interceptando los horarios de disponibilidad del profesor y el tipo de aula
- Materia $\mathcal{M}_i$: Cada grupo tiene asociado una materia que es la que va a ser dictada por el profesor $p_i$ (esta es programada cuando se crea el grupo)

Para cada $g_i$, la oficina de Admisiones y Registro define los valores de $t_i$, $C_i$, $p_i$ y $\mathcal{M}_i$. El valor inicial de $h_i$ se calcula a partir de la disponibilidad de $p_i$ y $t_i$ y el valor de $\mathcal{M}_i$. Una primera

diferencia de este modelo con lo propuesto en[46], es que en dicho modelo existía la noción de semestre perfecto como base de la planificación. Bajo esas circunstancias, todos los grupos que compartían el mismo número, debían ser compatibles entre sí. En la situación actual, que es para la que se diseña este modelo, no existe la noción del semestre perfecto y por esa razón, en la sección 6 se es necesario incorporar la información del registro de las materias que van a ser impartidas en el semestre siguiente a los estudiantes, como insumo para la planificación.

En cada materia $M_i$ se programa:

- Horas requeridas $r_i$: el número de horas que deben ser programadas semanalmente para que un grupo pueda recibir clase

Para cada profesor $p_i$ se tiene:

- Horario $h_{p_i}$: El cual refleja la disponibilidad horaria, es decir cuáles son las horas que tiene disponibles para dictar clase a la semana. El horario se va modificando a medida que al profesor se le programan grupos.

Para cada aula $a_i$ se tiene:

- Tipo de aula $t_i$: Las aulas se clasifican dependiendo de los recursos que tengan. Para efectos de la planificación, las aulas que tienen las mismas características se agrupan bajo un mismo tipo de aula
- Horario $h_{a_i}$: La horas en las que el aula no está ocupada y puede ser programada. Este horario cambia a medida que se programan grupos en el aula
- Capacidad $c_i$: La capacidad máxima del aula, es decir, el número máximo de alumnos que pueden ver una materia en ella

El objetivo del presente trabajo de grado es encontrar la combinación de profesores, aulas, grupos y horarios, que se acomoden de la mejor forma posible a los requerimientos de la Universidad EAFIT. Para hacer esto lo que inicialmente se necesita es encontrar cual de las anteriores variables restringe más la búsqueda.

Si se toma inicialmente a todos los profesores con sus restricciones de horarios y los grupos en donde estos pueden ser programados, siendo:

- $p_i$ = el profesor $i$
- $g_j$ = el grupo $j$

Entonces sea $G_{ij}$ el asignar el profesor $p_i$ al grupo $g_j$ de tal manera que el horario($h_{g_j}$) del grupo $g_j$ sea el horario ($h_{p_i}$) del profesor $p_i$, entonces el conjunto de grupos quedaría así:

- $\{G_{ij}/h_{g_j} \leftarrow h_{p_i}\}$

Ya que si hay dos profesores ($p_a$ y $p_b$) que pueden dictar la misma materia y pueden ser asignados a dos grupos ($g_i$ y $g_j$), se llega a la conclusión de que tener asignado $G_{ai}$ ó $G_{bi}$ tienen la misma relevancia, lo mismo que tener $G_{aj}$ ó $G_{bj}$, ya que el grupo no tiene nada asignado hasta el momento. Es decir no se produce ninguna diferencia al asignar un profesor a un grupo en específico, pues el horario que tiene el grupo, está restringido al horario del profesor, por lo que asignar un profesor a un grupo o a otro, tendría la misma validez. Este punto marca también una diferencia con respecto al modelo planteado en [46], donde la forma de asignar los profesores a los grupos afectaba el resultado de la programación. Para la planificación de

los grupos, el profesor solo provee el horario de disponibilidad, así que al referirse a $G_{ij}$, de manera abreviada se estará refiriendo a $g_j$.

### Selección del Grupo

En primer lugar, se define el coeficiente asociado a cada grupo, como la demanda que cada grupo tiene sobre cada hora. Sea $g_i$ un grupo, $r_i$ la intensidad horaria semanal para dictar la materia asociada al grupo $g_i$, o sea el número de horas semanales que requiere un profesor dictar la materia en el grupo, y $h_{g_i}$ es el horario de disponibilidad del grupo $g_i$, entonces se tiene que $h_{g_i}[j]$ es la disponibilidad que tiene el grupo $g_i$ en la hora $j$.

Sea la demanda $\mathcal{D}g_i$ que ejerce el grupo $g_i$ sobre cada hora disponible y $n$ el número de horas totales del horario, tenemos que:

$$\mathcal{D}_{g_i} = \frac{\sum_{j=1}^{n} h_{g_i}[j]}{r_i} \tag{5.2}$$

Cuando $\mathcal{D}_{g_i} \geq 1$, habrá posibilidades de que el grupo pueda ser programado. La tabla 5.1 resume estos hechos:

| | |
|---|---|
| $\mathcal{D}_g \geq 1$ | El grupo se puede programar |
| $\mathcal{D}_g < 1$ | El grupo no se puede programar |

**Cuadro 5.1:** Interpretación de $\mathcal{D}_g$

Es importante notar que $1/\mathcal{D}g_i$ puede ser interpretado como la probabilidad de que el grupo $g_i$ quede programado en una hora específica, tal y como se demuestra en el teorema 1.

Teorema 1. *Sea $g_i$ un grupo con coeficiente $\mathcal{D}_{g_i}$, entonces la probabilidad de que el grupo $g_i$ quede programado a la hora $h_{g_i}$ es:*

$$p(h_{g_i}) = \frac{1}{\mathcal{D}_{g_i}}$$

*Demostración.* Sea $r_i$ el número de horas requeridas para programar el grupo $g_i$ y $d_i$ el número de horas disponibles que tiene $g_i$ para ser programado, i.e. $d_i = \sum_{j=1}^{n} h_{g_i}[j]$, entonces el número total de posibles horarios para el grupo está dado por

$$N_o = \binom{d_i}{r_i} \tag{5.3}$$

puesto que de las $d_i$ horas disponibles solo se requieren $r_i$ para programar el grupo. La expresión (5.3) calcula el número total de combinaciones de $r_i$ elementos que puedo construir con $d_i$ elementos.

Ahora fijemos una hora $h_{g_i}$ tal que el grupo pueda ser programado a esa hora, y así el número total de horarios en los que se puede programar el grupo que no incluyan la hora $h_{g_i}$ está dado por

$$n_o = \binom{d_i - 1}{r_i} \tag{5.4}$$

Entonces, la probabilidad de que el grupo quede programado a la hora $h_{g_i}$ está dado por

$$p(h_{g_i}) = \frac{N_o - n_o}{N_o} \tag{5.5}$$

De (5.3) y (5.4) tenemos que:

$$\begin{aligned}
p(h_{g_i}) &= \frac{\binom{d_i}{r_i} - \binom{d_i - 1}{r_i}}{\binom{d_i}{r_i}} \\
&= \frac{\frac{d_i!}{r_i!(d_i - r_i)!} - \frac{(d_i-1)!}{r_i!((d_i-1)-r_i)!}}{\frac{d_i!}{r_i!(d_i-r_i)!}} \\
&= \frac{\frac{d_i!}{r_i!(d_i-r_i)!} - \frac{(d_i-1)!}{r_i!(d_i-r_i-1)!}}{\frac{d_i!}{r_i!(d_i-r_i)!}} \\
&= \frac{\frac{d_i(d_i-1)!}{r_i!(d_i-r_i)(d_i-r_i-1)!} - \frac{(d_i-1)!}{r_i!(d_i-r_i-1)!}}{\frac{d_i!}{r_i!(d_i-r_i)!}} \\
&= \frac{\frac{(d_i-1)!}{r_i!(d_i-r_i-1)!}\left(\frac{d_i}{d_i-r_i} - 1\right)}{\frac{d_i(d_i-1)!}{r_i!(d_i-r_i)(d_i-r_i-1)!}} \\
&= \frac{\frac{\cancel{(d_i-1)!}}{\cancel{r_i!(d_i-r_i-1)!}}\left(\frac{d_i}{d_i-r_i} - 1\right)}{\frac{d_i\cancel{(d_i-1)!}}{r_i!(d_i-r_i)\cancel{(d_i-r_i-1)!}}} \\
&= \frac{\frac{d_i}{d_i-r_i} - 1}{\frac{d_i}{d_i-r_i}} \\
&= \frac{\frac{d_i-(d_i-r_i))}{d_i-r_i}}{\frac{d_i}{d_i-r_i}} \\
&= \frac{\frac{d_i-(d_i-r_i))}{\cancel{d_i-r_i}}}{\frac{d_i}{\cancel{d_i-r_i}}}
\end{aligned}$$

$$
\begin{aligned}
&= \frac{d_i - (d_i - r_i)}{d_i} \\
&= \frac{d_i - d_i + r_i}{d_i} \\
&= \frac{\cancel{d_i} - \cancel{d_i} + r_i}{d_i} \\
&= \frac{r_i}{d_i}
\end{aligned}
$$

De donde se deduce que:

$$
p(h_{g_i}) = \frac{\binom{d_i}{r_i} - \binom{d_i - 1}{r_i}}{\binom{d_i}{r_i}} = \frac{r_i}{d_i} = \frac{r_i}{\sum_{j=1}^{n} h_{g_i}[j]} = \frac{1}{\mathcal{D}g_i}
$$

□

Ya que se sabe cuales son las restricciones a tener a la hora de buscar un grupo (5.1.3), se procede a buscar el grupo con la menor disponibilidad que sea mayor o igual a 1. Esto hará que los grupos más restringidos a la hora de dictar la clase sean los primeros en ser escogidos.

Sea *gg* un grupo tal que:

$$
\mathcal{D}_{gg} = \min_{\forall g_i \in G} \mathcal{D}g_i \tag{5.6}
$$

Entonces *gg* es el grupo más opcionado para ser programado por su restricción de horario, pues según el teorema 1, es el que mayor probabilidad tiene de ser programado en las horas

disponibles.

### Búsqueda del Horario

El siguiente paso es identificar la demanda que ejercen todos los grupos en cada hora. La urgencia $U$ de un grupo $g$, es la probabilidad de quedar programado y este es el cociente que hay entre la intensidad horaria necesaria para ser dictado el grupo y su disponibilidad. Sea $U_i$ la urgencia del grupo $g_i$, entonces tenemos que:

$$U_i = \frac{r_i}{\sum_{j=1}^{n} h_{gi}[j]} = \frac{1}{\mathcal{D}g_i} = p(h_{g_i}) \tag{5.7}$$

De donde se tiene que entre mayor sea la disponibilidad del grupo, menor será la urgencia, siendo $U_i = 1$ el máximo valor que podrá obtener la urgencia, que solo se dá cuando hay una alternativa (un horario) para programar el grupo.

Sea la matriz $\mathcal{G}$ del tamaño del horario, tal que en $\mathcal{G}[i]$ está el número de grupos no programados aún(libres), que pueden ser programados a la hora $i$:

$$\mathcal{G}[i] = \sum_{\forall g_j \in G} h_{g_j}[i] \tag{5.8}$$

Sea $\mathcal{U}$ la matriz de factor de urgencias, una matriz del mismo tamaño que el horario, donde la entrada $\mathcal{U}[h]$ de la matriz, contiene la suma de las urgencias de todos los grupos que pueden ser dictados a la hora $h$ sobre el número de grupos que pueden dictar las clases en esa misma

hora, lo que sería la esperanza de que el grupo quede programado en la hora $h$, la cual se define como:

$$\mathcal{U}[i] = \frac{\sum_{\forall g_j \in G} U_j * h_{g_j}[i]}{\mathcal{G}[i]} \tag{5.9}$$

Ahora, el horario más adecuado para dictar la clase de $gg$ será el horario que tenga el menor valor en la matriz de factor de urgencias $\mathcal{U}[h]$, es decir, donde el número esperado de grupos que pueden quedar programados sea el menor. Así, a medida que se vayan programando los diferentes grupos, se van asignado a los horarios menos demandados haciendo que no haya horario con sobrecarga de grupos programados.

Primero se haya el menor factor de urgencia $\mathcal{U}d$:

$$\mathcal{U}d = \min_{h=1}^{n} \mathcal{U}[h] \tag{5.10}$$

El horario a escoger será entonces aquel, que tenga el menor factor de urgencia. Sea $hh$ el horario donde $gg$ se programará. Sea $hh[i]$ una hora en específico, entonces se programará de tal manera que en las horas donde el factor de urgencia sea la menor, es 1, en otro caso es 0:

$$hh[i] = \begin{cases} 1 & \text{Si } \mathcal{U}d = \mathcal{U}[h_i] \\ 0 & \text{otro caso} \end{cases} \tag{5.11}$$

Así que el horario $hh$ será el horario escogido para programar el grupo $gg$.

### Asignación del Aula

Una vez escogido el grupo a programar($gg$) y el horario en el que va a ser programado($hh$), el siguiente paso es buscar un aula para dictar el curso.

Sea $a_i$ un aula y $ha_i$ el horario de disponibilidad del aula $i$ y $ha_i[j]$ es la disponibilidad que tiene el aula $i$ en la hora $j$, y $r_i$ el número de horas necesarias para dictar la materia que el grupo tiene asignada, tenemos que $\mathcal{D}a_i$ es la demanda que ejerce el aula $i$ sobre cada hora disponible cuando el aula tiene disponible el horario $hh$, por lo tanto se tiene que:

$$\mathcal{D}a_i = \begin{cases} \frac{\sum_{j=1}^{n}(ha_i[j])}{r_i} & \text{Si } \sum_{j=1}^{n}(ha_i[j] * hh[j]) \geq r_i \\ 0 & \text{otro caso} \end{cases} \tag{5.12}$$

Cuando el valor de $\mathcal{D}_{a_i} \geq 1$, la clase se dictará y el aula será programada. La cuadro 5.2 resume estos hechos:

| | |
|---|---|
| $\mathcal{D}_a \geq 1$ | El aula se puede programar |
| $\mathcal{D}_a < 1$ | El aula no se puede programar |

**Cuadro 5.2:** Interpretación de $\mathcal{D}_a$

Así pues se escoge el aula donde la demanda $\mathcal{D}a_i$ sea la mínima, es decir, donde el aula requiere menos horas, y de todas las que tengan la menor demanda se toma el aula más pequeña que tenga la capacidad para albergar a el grupo.

Sea *ag* el aula a escoger para el grupo *gg*, donde $C_g$ es el número de estudiantes que tiene el grupo y $t_g$ es tipo de aula que necesita el grupo para ser programado, y sea el aula $a_i$ tal que $c_i$ es la capacidad que tiene esta aula y $t_i$ el tipo de aula que tiene, entonces:

$$\mathcal{D}_{ag} = \min_{\forall a_i \in \mathcal{A}} \mathcal{D}a_i \mid t_g = t_i \text{ y } \min c_i > C_g \tag{5.13}$$

Por lo que *ag* es el aula con el horario disponible en el horario escogido para el grupo (*hh*), del mismo tipo de aula necesaria para el grupo y con la mínima capacidad suficiente para que todo el grupo esté en el aula.

Al finalizar este proceso, el grupo *gg* será asignado en el horario *hh* y en el aula *ag*. Se elimina el grupo *gg* de la lista de grupos que se deben programar y se modifica el horario del profesor y del aula, marcando estos horarios como no disponibles a la hora programada para poder cumplir con las restricciones duras. Finalmente, si aún quedan grupos por programar, se repite nuevamente el proceso.

### 5.2.1. Casos de excepción

Como el algoritmo implementado está basado un algoritmo *goloso-miope*, existen casos en donde es posible que seleccionando un orden diferente, podría entregar una solución con mejores resultados.

Sea el grupo $g_1$ con $\mathcal{D}g_1 = \frac{d_1}{r_1}$ y el grupo $g_2$ con $\mathcal{D}g_2 = \frac{d_2}{r_2}$ tal que $\frac{d_1}{r_1} < \frac{d_2}{r_2}$, por lo que en

este caso, de acuerdo con la ecuación 5.6, se programaría el grupo $g_1$ antes que $g_2$.

Entonces, podría existir un caso donde al programar primero $g_1$ sea imposible programar $g_2$ y, sin embargo, al programar primero $g_2$ se podría programar luego $g_1$, aunque se cumpla que

$$\frac{d_1}{r_1} < \frac{d_2}{r_2} \tag{5.14}$$

Sea $n \geq 0$ el número de horas usadas para programar $g_2$ y que son comunes con $g_1$, tal que :

$$\frac{d_2 - n}{r_2} < 1 \tag{5.15}$$

$$\frac{d_1 - n}{r_1} > 1 \tag{5.16}$$

Es decir, que a pesar de ser más probable para asignar $g_1$, sería mucho mejor asignar primero $g_2$.

- Entonces de 5.15 tenemos que:

$$\frac{d_2 - n}{r_2} < 1 \tag{5.17}$$

$$\frac{d_2}{r_2} - \frac{n}{r_2} < 1 \tag{5.18}$$

$$\frac{d_2}{r_2} < 1 + \frac{n}{r_2} \tag{5.19}$$

$$\frac{d_2}{r_2} - 1 < \frac{n}{r_2} \tag{5.20}$$

De 5.16 obtenemos:

$$\frac{d_1 - n}{r_1} > 1 \tag{5.21}$$

$$\frac{d_1}{r_1} - \frac{n}{r_1} > 1 \tag{5.22}$$

$$\frac{d_1}{r_1} > 1 + \frac{n}{r_1} \tag{5.23}$$

$$\frac{d_1}{r_1} - 1 > \frac{n}{r_1} \tag{5.24}$$

De 5.14 tenemos que

$$\frac{d_1}{r_1} < \frac{d_2}{r_2} \tag{5.25}$$

$$\frac{d_1}{r_1} - 1 < \frac{d_2}{r_2} - 1 \tag{5.26}$$

Así que de 5.20, 5.24, 5.26 tenemos:

$$\frac{n}{r_1} < \frac{d_1}{r_1} - 1 < \frac{d_2}{r_2} - 1 < \frac{n}{r_2} \tag{5.27}$$

$$\frac{n}{r_1} < \frac{n}{r_2} \tag{5.28}$$

$$r_2 < r_1 \tag{5.29}$$

$$\boxed{r_2 < r_1} \tag{5.30}$$

Por lo que dado el caso de que el número de horas requeridas en el grupo $g_1$ sea mayor que en $g_2$ y la demanda $\mathcal{D}g_1$ es menor que $\mathcal{D}g_2$, entonces podría ser posible mejor primero asignar primero $g_2$ y luego $g_1$.

- También de 5.15 se tiene que:

$$\frac{d_2 - n}{r_2} < 1 \tag{5.31}$$

$$d_2 - n < r_2 \tag{5.32}$$

$$d_2 - r_2 < n \tag{5.33}$$

y de 5.16:

$$\frac{d_1 - n}{r_1} > 1 \tag{5.34}$$

$$d_1 - n > r_1 \tag{5.35}$$

$$d_1 - r_1 > n \tag{5.36}$$

De 5.33 y 5.36 tenemos que:

$$\boxed{d_2 - r_2 < n < d_1 - r_1} \tag{5.37}$$

Por lo que la diferencia de horas entre las horas disponibles y las horas requeridas del grupo uno deben de ser mayores que las del grupo dos para poder asignar primero $g_2$.

### 5.2.2. Conclusiones

En este capítulo se mostró porqué el método escogido, el cual es una solución estocástica, es una buena opción para programar grupos, aulas y profesores para una universidad y en especial para la Universidad EAFIT, pues cumple con los requerimientos que se piden. También se mostró que la solución dada no es una solución óptima, ya que pueden existir casos en donde bajo ciertos criterios pueda obtenerse una solución mucho mejor que la entregada.

*El futuro tiene muchos nombres. Para los débiles es lo inalcanzable. Para los temerosos, lo desconocido. Para los valientes es la oportunidad.*

Victor Hugo

# 6

# Implementación de la planificación de horarios para la Universidad EAFIT

Para la implementación de la solución del problema de la planificación de horarios en la Universidad de EAFIT, se utilizó el lenguaje de programación *C++*.

Se implementaron dos algoritmos, uno secuencial y otro en paralelo con MPI y se habla de

la implementación de cada uno, sus características y los problemas encontrados en el camino.

Los algoritmos desarrollados en este trabajo, entregan una lista de grupos programados a partir de la evaluación de una lista de grupos disponibles (sin programar), profesores con sus horarios de disponibilidad y materias a dictar, aulas con su disponibilidad y tamaño, una lista alumnos con la lista de horario de clase y materias registradas, y una lista de materias con el número de horas que necesita para ser dictadas.

## 6.1. Algoritmo Secuencial

Para la implementación del algoritmo secuencial solo se uso el lenguaje de programación *C++*. A continuación se muestra de manera general el algoritmo y de las funciones que lo conforman.

El algoritmo 1 busca la combinación de grupos, horarios y aulas que tengan la mejor disposición en el momento de ser programados.

El algoritmo 2, es una función que en el momento de buscar el grupo más opcionado, hace un análisis de todos los horarios de los grupo, y de la disponibilidad que tienen(horas disponibles que tiene el profesor) vs las horas que se requieren para dictar la materia de cada grupo, entrega el grupo que tenga menor proporción mayor que 1, ya que los que tengan esa proporción menor, no tienen la cantidad de horas para ser dictado.

En el algoritmo 3 se describe la forma de buscar el horario para el grupo más opcionado.

Algoritmo 1: Algoritmo que busca la mejor solución para la planificación de horarios para la universidad de EAFIT

```
Entrada : Lista de grupos disponibles G, Lista de grupos programados GP, profesores
          P, aulas A, alumnos Al
Salida  : Lista de grupos programados
1  asignarProfesoresAGrupos(P,G); ▷ Asigna a cada grupo un profesor
2  while G ≠ ∅ do
3      grupo ← traerGrupoDg(G);
4      horario ← traerHorarioDg(grupo);
5      aula ← entregaAula(grupo, horario);
6      grupo.programar(horario,aula);
7      GP.agregar(grupo);                ▷ Guarda el grupo en programado
8      P⟨grupo.profesor⟩.asignarHorario(horario);
9      A.asignarHorario(horario);
10     G.remover(grupo);                 ▷ Elimina el grupo de la lista de
        disponibles
11 end
```

Algoritmo 2: Función que entrega el grupo mas opcionado para programar

```
Entrada : Lista de grupos Gl que aún no se han programado
Salida  : El grupo con menor demanda
1  traerGrupoDg(Grupos Gl)
2  Gg ← null;
3  dg ← 24*7;
4  for g ∈ Gl do
5      dgt ←Sum(g.horasDisponibles⟨⟩)/g.horasRequeridas;
6      if dgt ⩾ 1 and dg > dgt then
7          dg ←dgt;
8          Gg ← g
9      end
10 end
11 return Gg
```

Algoritmo 3: Función que entrega el mejor horario para programar el grupo

```
   Entrada : Grupo gl con la disponibilidad para dar clase
   Salida  : El horario en el que se programarán las clases
 1 traerHorarioDg(Grupo gl);          ▷ n=7*24*2 Número de periodos a la
     semana
 2 fl ⟨a_1, a_2, ..., a_n⟩ ;                                   ▷ a_i=0
 3 for i ← 1 to n do
 4 |   urg ←0;
 5 |   ng ←0;
 6 |   for g ∈ Grupos do
 7 |   |   urg ← g.horasRequeridas/Sum(g.horasDisponibles⟨⟩);
 8 |   |   fl ⟨a_i⟩ ←fl ⟨a_i⟩ + g.horasDisponibles⟨i⟩*urg;
 9 |   |   ng ←ng +g.horasDisponibles⟨i⟩;
10 |   end
11 |   fl ⟨a_i⟩ ← fl⟨a_i⟩ / ng
12 end
13 horario = combinaHorario(gl,fl)  return horario
```

Para ello, se suman la probabilidad que cada grupo tiene para ser programado en cada horario, dividido el número de grupos que pueden ser dictados en cada horario(Vector de urgencias de los grupos). Los valores mayores son los horarios en donde hay más posibilidades de que un grupo se pueda programar, y los valores menores, son los horarios que tienen más dificultad para ser programado. Con algoritmo 4 se retorna el mejor horario para el grupo más opcionado.

*combinaHorario*, el algoritmo 4 evalúa a partir de las horas disponibles del grupo más opcionado, y de las horas requeridas para ser programado la combinación horaria tal que los valores del vector de urgencias sean los menores.

Algoritmo 4: Función que busca a partir del horario del grupo *gl* y el vector de urgencias de los grupos fl, el horacombinaHorariocombinaHorariorio con menor urgencia para dictar la clase

Entrada : Grupo *gl* con la disponibilidad para dar clase
Salida : El horario en el que se programarán las clases

```
1  combinaHorario(Grupo gl, Float fl ⟨⟩)
2  ps ← 0;
3  pos ← 0;
4  for i ← 1 to n do
5  |   pst ← 0;
6  |   for j ← i to i + gl.horasRequeridas do
7  |   |   pst ← fl ⟨j⟩;
8  |   end
9  |   if ps > pst then
10 |   |   pos ← i;
11 |   end
12 end
13 for i ← 1 to n do
14 |   horario ⟨i⟩ ← 1;
15 end
16 return horario
```

Algoritmo 5: Función que entrega el aula para el grupo escogido

```
   Entrada : Grupo gl con la disponibilidad para dar clase
   Entrada : Horario h horario de programación de las clases
   Salida  : Aula más opcionada para dar clases
 1 entregaAula(Grupo gl, Horario h ⟨⟩)
 2 Au ← null;
 3 da ← 24*7;
 4 for a ∈ Aulas do
 5 |   for i ← 1 to n do
 6 |   |   dat ← dat + a.horasDisponibles⟨i⟩ ∗ h⟨i⟩;
 7 |   end
 8 |   if dat ≥ gl.horasRequeridas then
 9 |   |   dat ← sum(a.horasDisponibles⟨⟩) / gl.horasRequeridas;
10 |   end
11 |   if dat ⩾ 1 and da > dat and gl.tipoAula = a.tipoAula then
12 |   |   da ←dat;
13 |   |   Au ← g
14 |   end
15 end
16 return Au
```

Para saber cual es el aula para el grupo más opcionado en el algoritmo 5 se busca cual es aula más pequeña que no ha sido programada en el horario escogido para programar el grupo y que tenga el mismo tipo de aula requerida por el grupo.

## 6.2. Algoritmo Paralelo

En la implementación del algoritmo paralelo junto con *C++* se uso *MPI*. El algoritmo como tal, tiene la misma funcionalidad del algoritmo secuencial, pero con modificaciones en algunas funciones, con el fin de hacer la paralelización. Como la paralelización que se va a hacer es a nivel de procesos, lo usual cuando hay este tipo de soluciones es hacer que uno de los procesos sea el proceso maestro, el cual dirige el comportamiento de el resto de procesos(esclavos):

En el algoritmo asignarProfesoresAGrupos, algoritmos 6 y 7 lo que se hacen es que la carga de los grupos se divide entre el número de procesadores que hay, haciendo que cada uno de ellos procese una parte, así en recibirDgGrupos() devuelve el grupo más opcionado para ser programado encontrado por cada procesador, el procesador maestro escoge mejor de ellos y buscando el mejor horario para que este sea programado. Luego envía a todos los procesadores el grupo y el horario para que calculen el aula mas opcionada. Cada procesador devolverá el aula y el procesador maestro evaluará cual debe ser el aula a escoger. Este proceso se hará hasta que haya terminado de programar todos los grupos.

Algoritmo 6: Algoritmo paralelo de la planificación de horarios, proceso maestro

Entrada : Lista de grupos disponibles $G$, Lista de grupos programados $GP$, profesores $P$, aulas $\mathcal{A}$, alumnos $\mathcal{A}l$, *core* número del procesador donde se ejecuta el algoritmo

Salida : Lista de grupos programados

```
1  asignarProfesoresAGrupos(P,G); ▷ Asigna a cada grupo un profesor
2  while G ≠ ∅ do
3      grupo ←recibirDgGrupos();  ▷ Trae el coheficiente mas pequeño
        de cada esclavo y devuelve el grupo con el menor
        coheficiente
4      enviarAEsclavosGrupo(grupo);
5      horario ← traerHorarioDg(grupo);  ▷ No se paraleliza la búsqueda
        de horario
6      enviarAEsclavosHorario(horario);
7      aula ←recibirDgAula();      ▷ Trae el coheficiente mas de cada
        esclavo y devuelve el aula con el mejor coheficiente
8      grupo.programar(horario,aula);
9      GP.agregar(grupo);              ▷ Guarda el grupo en programado
10     P⟨grupo.profesor⟩.asignarHorario(horario);
11     A.asignarHorario(horario);
12     G.remover(grupo);            ▷ Elimina el grupo de la lista de
        disponibles
13 end
```

Algoritmo 7: Algoritmo paralelo de la planificación de horarios, Otros procesos(esclavos)

Entrada : Lista de grupos disponibles $G$, Lista de grupos programados $GP$, profesores $P$, aulas $\mathcal{A}$, alumnos $\mathcal{A}l$, *core* número del procesador donde se ejecuta el algoritmo

Salida : Lista de grupos programados

```
1  asignarProfesoresAGrupos(P,G); ▷ Asigna a cada grupo un profesor
2  while G ≠ ∅ do
3  |   grupo, dg ← traerGrupoDgParalelo(G, core);     ▷ cada core calcula
   |    una parte de los grupos y entrega el menor de ellos
4  |   enviarAMaestro(grupo, dg);
5  |   grupo ←recibirGrupoMaestro();
6  |   horario ←recibirHorarioMaestro();
7  |   aula ← entregaAula(grupo, horario, core);     ▷ cada core calcula una
   |    parte de los aulas y entrega la mejor aula
8  |   enviarAMaestro(aula);
9  |   G.remover(grupo);            ▷ Elimina el grupo de la lista de
   |    disponibles
10 end
```

*El crecimiento es un proceso de prueba y error: es una experimentación. Los experimentos fallidos forman parte del proceso en igual medida que el experimento que funciona bien.*

Benjamin Franklin

# 7

# Conjunto de experimentos: Planificación de horarios en la Universidad EAFIT

La solución entregada por el algoritmo, cumple con las exigencias y con las necesidades impuesta por la universidad EAFIT. A continuación se muestran el conjunto de datos usados y los resultados obtenidos por el algoritmo.

Para el caso probado, se tomaron los datos de la planificación de horarios del año 2013, semestre 1. En este semestre el conjunto de valores fue:

- 2568 Grupos Matriculados
- 3237 Materias
- 1227 Profesores
- 434 Aulas
- 9535 Estudiantes

Para poder medir el tiempo computacional, todas las pruebas fueron realizadas en un mismo equipo que cuenta con las características:

| | |
|---|---|
| Sistema operativo | Linux Arch |
| Procesador | Intel Core i7(3.10GHz), cuadcore, $3^{ra}$ generación |
| Memoria | 8GB de Ram |
| Disco Duro | 32GB de SSD |

## 7.1. Casos de prueba

Para cada una de las pruebas hechas, se usó la herramienta *time* de *GNU/Linux*, esta herramienta mide el tiempo de ejecución de un programa, el porcentaje de uso de CPU y la memoria usada durante la ejecución.

### 7.1.1. Pruebas Algoritmo Secuencial

Para el primer caso, las pruebas hechas con el algoritmo secuencial (Sección 6.1), dieron como resultado:

```
El numero de profesores disponibles es: 1227

El numero de materias disponibles es: 3237

El numero de aulas disponibles es: 434

El numero de grupos disponibles es: 2568


Resultados:

Grupos asignados: 2435

15.40 user

0.00 system

0:15.41 elapsed

99% CPU
```

Del conjunto de datos suministrados por la universidad, hay un total de 2568 grupos a programar, y de estos al terminar la ejecución, se fueron programados 2435, por lo que el porcentaje de grupos asignados es de 94.82 %. La ejecución de la asignación de los grupos fue realizada en 15.40 segundos (El valor de 99 % es el uso de procesador durante la ejecución, en este caso el procesador tuvo un 99 % de uso durante la ejecución del algoritmo. En la programación paralela es usual encontrar porcentajes mayores al 100 %, esto es debido a que se suma el valor de uso de cada uno de los procesadores, así si se tiene que hay un uso del 395 % de uso

con 4 procesadores, en promedio hay un 98.7 % de uso por procesador).

En el cuadro 7.1 se muestra como queda la distribución del número de grupos programados por fracción de tiempo(30 minutos). En la asignación se tiene que de lunes a viernes hay un promedio de 43,5 grupos programados por fracción de tiempo, con una desviación estándar de 3,6, siendo 51, el número máximo de grupos programados y de 34 el horario con menos.

Notas importantes:

- En la implementación, los grupos se pueden programa de las siguientes maneras: Si se tiene una materia de un grupo de dos horas para ser dictada, se programa de manera consecutiva, es decir, en un solo bloque. Si se tiene una materia de tres horas, cuatro horas o seis horas requeridas para ser dictada, se buscará programarla en franjas de hora y media para los bloque de tres horas, y dos horas para el resto, distanciados por un día intermedio a la misma hora. Esto con el fin de que no haya bloques de clase muy largos.
- Hay que tener en cuenta que cuando se hizo la prueba, aparece que hay 434 aulas en la universidad, por lo que solo estaría ocupada el 10 % de las aulas, lo que haría que estuviese la mayor parte del tiempo la universidad sola. Haciendo un análisis mas detallado de las aulas, se encontró que el concepto de aula no es solo el lugar en donde se puede dictar una clase. Se encontró que las canchas de fútbol, baloncesto y tenis de campo, las salas de reunión de algunos bloques, salones de desarrollo artístico, salas de dibujo y de música, aula móvil, aulas de idiomas, cubículos de la biblioteca, salas de audición y multimedia, los auditorios y algunas aulas especiales son catalogadas como aulas. Así que haciendo un análisis más detallado sobre la cantidad de aulas que realmente usa el algoritmo para programar es de 72 aulas disponibles para programar. Al mirar el cuadro 7.1 y tomando como referencia los horarios de los días de lunes a viernes en horas de clase, se llegó a los resultados de un máximo de uso del 70,8 %(lunes y miercoles con 51 grupos asignados) y un mínimo del 47,3 %(viernes con 34 grupos asignados).

| Hora | Lunes | Martes | Miércoles | Jueves | Viernes | Sábado |
|---|---|---|---|---|---|---|
| 6:00-6:30 | 51 | 44 | 45 | 45 | 40 | 4 |
| 6:30-7:00 | 51 | 44 | 45 | 45 | 40 | 4 |
| 7:00-7:30 | 50 | 46 | 44 | 45 | 42 | 3 |
| 7:30-8:00 | 45 | 46 | 45 | 45 | 41 | 1 |
| 8:00-8:30 | 47 | 46 | 50 | 44 | 40 | 25 |
| 8:30-9:00 | 46 | 45 | 44 | 39 | 38 | 25 |
| 9:00-9:30 | 45 | 45 | 49 | 47 | 45 | 24 |
| 9:30-10:00 | 44 | 44 | 48 | 49 | 45 | 23 |
| 10:00-10:30 | 44 | 47 | 45 | 45 | 41 | 28 |
| 10:30-11:00 | 47 | 46 | 49 | 48 | 43 | 29 |
| 11:00-11:30 | 47 | 44 | 45 | 43 | 39 | 23 |
| 11:30-12:00 | 46 | 43 | 39 | 37 | 35 | 20 |
| 12:00-12:30 | 45 | 45 | 49 | 43 | 39 | 12 |
| 12:30-13:00 | 43 | 43 | 50 | 41 | 38 | 9 |
| 13:00-13:30 | 46 | 45 | 43 | 39 | 38 | 5 |
| 13:30-14:00 | 43 | 46 | 47 | (A1:A150) 40 | 42 | 3 |
| 14:00-14:30 | 43 | 46 | 44 | 42 | 43 | 6 |
| 14:30-15:00 | 43 | 45 | 42 | 37 | 42 | 5 |
| 15:00-15:30 | 44 | 44 | 45 | 42 | 36 | 3 |
| 15:30-16:00 | 44 | 43 | 44 | 42 | 35 | 3 |
| 16:00-16:30 | 46 | 47 | 44 | 41 | 40 | 4 |
| 16:30-17:00 | 47 | 47 | 46 | 40 | 38 | 1 |
| 17:00-17:30 | 48 | 47 | 46 | 42 | 41 | 5 |
| 17:30-18:00 | 44 | 45 | 41 | 38 | 40 | 4 |
| 18:00-18:30 | 45 | 44 | 48 | 39 | 39 | 1 |
| 18:30-19:00 | 44 | 45 | 47 | 40 | 38 | 0 |
| 19:00-19:30 | 45 | 45 | 46 | 37 | 40 | 0 |
| 19:30-20:00 | 48 | 47 | 51 | 44 | 46 | 0 |
| 20:00-20:30 | 46 | 45 | 43 | 40 | 34 | 0 |
| 20:30-21:00 | 43 | 42 | 38 | 35 | 34 | 0 |

**Cuadro 7.1:** Grupos programados por hora

- El número de materias que aparece en la implementación es muy superior al del número de grupos a programar, esto es debido a que la lista de las materias suministradas por Admisiones y Registro, están las materias de pregrado, postrado, maestría y cursos de extensión, además de materias que no pueden ser programadas por no contar con un profesor que pueda dictarlas.
- Para la programación de los grupos se toma en cuenta la disponibilidad de los profesores, en los datos suministrados por Admisiones y Registro se encuentran la disponibilidad de algunos profesores en la tarde de los sábados, así que es por esta razón que aparecen algunos grupos programados en este horario.

### 7.1.2. Pruebas Algoritmo Paralelo

Para la paralelización del algoritmo de asignación de horarios, se uso programación paralela, y se uso MPI para la paralelización. Los resultados obtenidos fueron:

- Con 2 cores los resultados fueron:

```
El numero de profesores disponibles es: 1227
El numero de materias disponibles es: 3237
El numero de aulas disponibles es: 434
El numero de grupos disponibles es: 2568

Resultados:
Grupos asignados: 2435
32.30 user
0.05 system
0:16.21 elapsed
199% CPU
```

- Para 3 cores los resultados fueron:

```
El numero de profesores disponibles es: 1227
El numero de materias disponibles es: 3237
El numero de aulas disponibles es: 434
El numero de grupos disponibles es: 2568

Resultados:
Grupos asignados: 2435
34.47 user
27.85 system
0:20.88 elapsed
298% CPU
```

- Para 4 cores los resultados obtenidos fueron:

```
El numero de profesores disponibles es: 1227
El numero de materias disponibles es: 3237
El numero de aulas disponibles es: 434
El numero de grupos disponibles es: 2568

Resultados:
Grupos asignados: 2435
49.04 user
51.96 system
0:25.55 elapsed
395% CPU
```

Como se vé en los resultados, a medida que se aumentó el número de cores(el número de cores, es el número de procesadores que están ejecutando un programa en el momento), el

tiempo de ejecución fue aumentando(2 cores: 16.21 seg, 3 cores: 20.28 seg, 4 cores: 25.55 seg), por lo que los resultados no concordaban con la teoría, pues a medida que hay más procesos de ejecución, el tiempo de ejecución debería de ser menor. Ante los resultados obtenidos, se hizo un análisis de las razones por la que los tiempos de ejecución iban incrementando. Se uso la herramienta *hprof* que es una herramienta para el análisis de ejecución de programas. Se encontró que en el momento de la carga de los datos, que se hace leyendo la información desde el disco duro, era la única parte del programa que estaba incrementando su ejecución, se muestra a continuación el porcentaje de tiempo que toma para cargar la información en la ejecución del algoritmo.

- Para 2 cores: 7.7 %
- Para 3 cores: 12.4 %
- Para 4 cores: 50 %

Así que se encontró que el proceso que está haciendo que la ejecución fuese más lenta a medida que se le agregan más cores, es la carga de los datos de los archivos, pues cada uno de los cores tiene que leer la información desde el disco duro, y no puede ser leída paralelamente, sino de forma secuencial, es decir una vez que un core está leyendo los datos del disco duro, los demás cores tienen que esperar a que el core que lee termine, luego de que termine solo uno de ellos continuará con la lectura de disco duro y de la carga de datos, y así sucesivamente hasta que todos los lo hagan.

Ante la imposibilidad de poder bajar el tiempo de la lectura de datos en el disco duro, se hizo un análisis de la ejecución del algoritmo y se restó el tiempo de lectura de disco duro, los resultados obtenidos fueron:

- Para 2 cores: 6.89437e-05 seg
- Para 3 cores: 3.9789e-05 seg
- Para 4 cores: 2.77731e-05 seg

Con lo cual se muestra que efectivamente el algoritmo si puede ser paralelizado, pero para que la paralelización sea efectiva, deberá tener una carga de datos significativamente grande, con el fin de que la carga de datos desde disco duro comparada con la ejecución de los datos sea aceptable.

## 7.2. Conclusiones

Con los datos suministrados por la Universidad EAFIT se pudieron hacer pruebas. Se encontró que los resultados obtenidos son muy buenos, con una asignación de los grupos por encima del 94 %. Con el desarrollo del algoritmo también se encontró que no hay sobrecarga de la asignación de grupos en los horarios y tampoco una sobrecarga en la asignación de aulas.

Se encontró también que al usar la programación paralela, los tiempo de respuesta eran más altos que los encontrados al usar programación secuencial, pero al analizar las ejecuciones de los programas se encontró que esto era debido a que la carga de datos desde el disco duro hacia el programa lleva mucho tiempo y esto lo convierte en un cuello de botella, pues

a pesar de contar con la programación paralela, la lectura a disco duro solo puede ser hecha por un core al tiempo, sin embargo con el análisis de la ejecución en parelelo y secuencial sin tomar en cuenta la carga de datos, si hay una paralelización en la ejecución de las funciones del algoritmo.

*La palabra tiene mucho de aritmética: divide cuando se utiliza como navaja, para lesionar; resta cuando se usa con ligereza para censurar; suma cuando se emplea para dialogar, y multiplica cuando se da con generosidad para servir.*

Carlos Siller

# 8

# Conclusiones

Uno de los problemas más grandes que tienen las universidades actualmente es la planificación los profesores en grupos y aulas, por este motivo se planteó desarrollar un algoritmo que permitiese hallar una solución muy buena a este problema.

Como la Universidad de EAFIT cuenta con APOLO, que es un centro de computación científico, se planteó poder implementar la solución paralelizándola y haciendo uso de los

recursos del centro de computo APOLO. Pero ante los resultados de la pruebas, se llegó a la conclusión de que la paralelización no ayuda en la mejora de los resultados, porque en la carga de archivos toma mucho tiempo y esta tarea no puede ser paralelizable, ya que la lectura a disco duro solo puede ser hecha por un procesador al tiempo.

Mientras se estuvo realizando el proyecto de grado se encontraron las siguientes conclusiones:

- El método que se usa para hallar la solución, es un método estocástico, que es una buena opción a la hora de programar los grupos, aulas y profesores.
- La solución presentada en el proyecto es una solución que cumple con los requerimientos exigidos por la Universidad EAFIT.
- La solución que se entregada, no es una solución óptima, y puede existir casos en donde la solución sea mejor que la entregada.
- El tiempo de búsqueda de la solución es entregado en un tiempo pertinente para las necesidades de la Universidad.
- Se encontró que a pesar de utilizar programación paralela, los tiempo de respuesta no eran los esperados, debido a que la carga de datos a memoria era muy demorado y el número de datos usados no eran lo suficientemente alto para que mejorara en los tiempos de respuesta.

*Yo, al igual que Dios, ni juego al azar ni creo en la casua-*

*lidad.*

V de Vendetta

# 9

# Trabajos Futuros

- Implementar una interfaz de usuario, para que sea fácil su interacción y así pueda no solo ser usado en la Universidad EAFIT, sino en otras universidades. También cabe la posibilidad de que en vez de solo crear una interfaz de usuario, se pueda crear un servicio web para que tenga mayor accesibilidad y visibilidad.
- Hacer una planificación en donde tenga en cuenta la programación de clases de un profesor con horarios de clase continuos, en donde se programe las aulas a la menor distancia posible.
- Buscar una planificación que tenga en cuenta que para las clases de los estudiantes que

tengan clases similares, haya la menor cantidad de tiempo entre clases.

- Hacer una programación de backtracking sobre la implementación del algoritmo, ya que como este fue hecho utilizando algoritmos voraces, es posible que se pueda encontrar mejores a las planteadas en el presente trabajo.

# Bibliografía


[1] Instituciones con acreditación de alta calidad. Accedida: 2016-10-01, https://www.icetex.gov.co/dnnpro5/es-co/cr

[2] International timetabling competition. http://sferics.idsia.ch/Files/ttcomp2002/.

[3] International timetabling competition. Accedida: 2015-09-03, https://www.utwente.nl/ctit/hstt/itc2011/welcome/.

[4] Open MPI: Open Source High Performance Computing, 2016. Accedida: 2016-12-11, https://www.open-mpi.org/.

[5] Kenneth R. Baker. *Introduction to sequencing and scheduling*. John Wiley & Sons, 1974.

[6] Ruggero Bellio, Luca Di Gaspero, and Andrea Schaerf. Design and statistical analysis of a hybrid local search algorithm for Course Timetabling. *Journal of Scheduling*, 15:49–61, 2012.

[7] Camille Beyrouthy and Edmund K. Burke. Clustering within timetabling conflict graphs. In *proceedings of the 3rd Multidisciplinary International Conference on Scheduling : Theory and Applications*, pages 553–556, 2007.

[8] J. F. Blakesley. Automation in college management. *College and University Business*, 27:39–44, 1959.

[9] RL Bracho. El problema del conjunto independiente en la selección de horarios de cursos. *Revista de Matemática: Teoría y Aplicaciones*, 10:156–167, 2009.

[10] E K Burke and J D Landa-Silva. The Design of Memetic Algorithms for Scheduling and Timetabling Problems. *Recent Advances in Memetic Algorithms*, Volume 166:289–311, 2004.

[11] Edmund Burke, Kirk Jackson, Jeffrey H. Kingston, and Rupert Weare. Automated University Timetabling: The State of the Art. *The Computer Journal*, 40(9):565–571, 1997.

[12] R Buyya. High Performance Cluster Computing: Architectures and Systems, Volume 1. *Prentice Hall PTR*, 82(Journal Article):327–350, 1999.

[13] Marco P. CARRASCO and Margarida V. PATO. A multiobjective genetic algorithm for the class/teacher timetabling problem. *Lecture notes in computer science*, pages 3–17, 2001.

[14] A. Ceder. *Public Transit Planning and Operation: Modeling, Practice and Behavior, Second Edition*. CRC Press, 2016.

[15] Ruey-Maw Chen and Hsiao-Fang Shih. Solving University Course Timetabling Problems Using Constriction Particle Swarm Optimization with Local Search. *Algorithms*, 6(2):227–244, 2013.

[16] Tim B Cooper and Jeffrey H. Kingston. The Complexity of Timetable Construction Problems. Technical Report 495, University of Sydney, 1995.

[17] M Coral, Y Jáuregui, and David Mauricio. Una solución al problema de cursos basado en heurísticas para el caso FISCT. *eventos.spc.org.pe*.

[18] Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein. *Introduction to Algorithms*, volume 25. 2009.

[19] Dave Corne, Peter Ross, and Hsiao-Lan Fang. Evolving Timetables. In *Practical Handbook of Genetic algorithms: Applications, Volume I*, pages 219—-276. 1995.

[20] S Daskalaki and T Birbas. Efficient solutions for a university timetabling problem through integer programming. *European Journal of Operational Research*, 160(1):106–120, 2005.

[21] Henry J. Davis. *An application of heuristics in management data processing; The scheduling problem*. PhD thesis, United States Naval Postgraduate School, 1962.

[22] D. de Werra. An introduction to timetabling. *European Journal of Operational Research*, 19(2):151–162, 1985.

[23] P Demeester and B Bilgin. A hyperheuristic approach to examination timetabling problems: benchmarks and a new problem from practice. *Journal of scheduling*, 2012.

[24] Edsger W. Dijkstra, C. A. R. Hoare, Ole-Jahn Dahl, Edsger W. Dijkstra, and C. A. R. Hoare. *Structured Programming*, volume 244. 1972.

[25] John J Dinkel, John Mote, and M A Venkataramanan. An Efficient Decision Support System for Academic Course Scheduling. *Operations Research*, 37(6):853–864, 1989.

[26] Emanuel Tellez Enriquez. Uso de una Colonia de Hormigas para resolver Problemas de Programacion de Horarios. Master's thesis, 2007.

[27] S. Even, A. Itai, and A. Shamir. On the complexity of time table and multi-commodity flow problems. *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*, 1975.

[28] JA Ospina Falla. Solución de un problema de asignación de horarios de mantenimiento en máquinas, equipos e instalaciones a la empresa Powel Continental Ltda., mediante la programación. Master's thesis, 2004.

[29] H. Fang. *Genetic Algorithms in Timetabling and Scheduling*. PhD thesis, University of Edinburgh, 1994.

[30] Mohammad Reza Feizi-Derakhshi and Mahdi Zandi. A solution to the problem of university examinations timetabling through information related to the units taken previous terms. In *2010 The 2nd International Conference on Computer and Automation Engineering, ICCAE 2010*, volume 1, pages 559–562, 2010.

[31] Michael J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, C-21(9):948–960, 1972.

[32] M R Garey and D S Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences). *Computers and Intractability*, page 340, 1979.

[33] G a Geist, J a Kohl, and P M Papadopoulos. PVM and MPI: a Comparison of Features. *Calculateurs Paralleles*, pages 137–150, 1996.

[34] Marcel Goic F and Carlos Caballero V. Aplicación de Algoritmos Genéticos para el mejoramiento del proceso de programación del rodaje en la industria del cine independiente. pages 1–19, 2005.

[35] Gruber & Petters GmbH. Untis. Accedida: 2014-02-04, http://www.grupet.at/.

[36] Grupo CF Developer. DocCF, Software de Gestión Escolar. Accedida: 2014-02-04, http://www.grupocfdeveloper.com.

[37] Shoshana Hahn-Goldberg. Defining, Modeling, and Solving a Real University Course Timetabling Problem. Master's thesis, University of Toronto, 2007.

[38] Martin Henz. *Constraint-based timetabling a case study*. PhD thesis, Charles University in Prague, 1996.

[39] A. Hertz. Tabu search for large scale timetabling problems. *European Journal of Operational Research*, 54(1):39–47, sep 1991.

[40] Omar J. Ibarra-Rojas and Yasmin A. Rios-Solis. Synchronization of bus timetabling. *Transportation Research Part B: Methodological*, 46(5):599–614, 2012.

[41] David S Johnson. Timetabling University Examinations. *The Journal of the Operational Research Society*, 41(1):39–47, 1990.

[42] Soria Alcaraz Jorge A., Carpio Valadez J Martin, and Terashima Marin Hugo. Academic timetabling design using hyper-heuristics. *Studies in Computational Intelligence*, 318:43–56, 2010.

[43] David Karger, Joel Wien, and Cliff Stein. Scheduling algorithms. Technical report, MIT; DartmouthCollege; Polytechnic University, 2007.

[44] Mohammed Azmi Al-Betar Khader, Ahamad Tajudin, and Taufiq Abdul Gani. A harmony search algorithm for university course timetabling. *Annals of Operations Research*, 2012.

[45] Gerald Lach and Marco E. Lübbecke. Curriculum based course timetabling: New solutions to Udine benchmark instances. *Annals of Operations Research*, 194(1):255–272, 2012.

[46] Juan G. Lalinde-Pulido and Bertha L. Velasquez-Serrano. Sistema de Programación Académica, 1992. Trabajo de grado, Universidad EAFIT.

[47] Gilbert Laporte and Sylvain Desroches. Examination timetabling by computer. *Computers and Operations Research*, 11(4):351–360, 1984.

[48] N L Lawrie. An integer linear programming model of a school timetabling problem. *The Computer Journal*, 12(4):307–316, 1969.

[49] Rhydian Marc Rhys Lewis. *Metaheuristics for University Course Timetabling*. PhD thesis.

[50] John Lions. Matrix reduction using the hungarian method for the generation of school timetables. *Commun. ACM*, 9(5):349–354, 1966.

[51] Lucero de Montserrat Ortiz_Aguilar, Juan Martín Carpio_Valadez, Héctor José Puga_Soberanes, Claudia Leticia Díaz_González, Carlos Lino_Ramírez, and Jorge Alberto Soria_Alcaraz. Comparativa de algoritmos bioinspirados aplicados al problema de calendarización de horarios. *Research in Computing Science*, 94:33–43, 2015.

[52] Armin Lüer and Jaime Bustos. Laboratorios de computación como cluster HPC. In *Workshop Internacional EIG2009*, 2009.

[53] Csermelyi Balogh Maria Edith. *On Scheduling Algorithms*. PhD thesis, Oregon State University, 1962.

[54] Miguel Mata-Pérez. Introducción a la programación lineal y entera Una simple presentación Programación lineal. Technical report, Universidad Autónoma de Nuevo León, Nuevo León, 2014.

[55] Richard H. McClure and Charles E. Wells. a Mathematical Programming Model for Faculty Course Assignments. *Decision Sciences*, 15(3):409–420, 1984.

[56] Ricardo Medel, César Martínez Spessot, and Julio Castillo. Hacia la Aplicación de la Computación de Alto Desempeño al Entorno Productivo Local. In *XIV Workshop de Investigadores en Ciencias de la Computación*, pages 750–754, 2012.

[57] José Mejía and Carlos Paternina. Asignación de horarios de clases universitarias mediante algoritmos evolutivos. *Revista Educación en Ingeniería*, 9:140–149, 2010.

[58] Elmer Freddy Mendoza-Apaza. *Generación y selección de horarios mediante algoritmos genéticos*. PhD thesis, Universidad mayor de San Andrés, 2009.

[59] Harlan D Mills. *Mathematical Foundations for Structured Programming*. 1972. Gaithersburg, MD: Federal Systems Division, Tennessee University.

[60] Angelo Monfroglio. Timetabling through a deductive database: A case study. *Data and Knowledge Engineering*, 3(1):1–27, 1988.

[61] Thomas Morton and David W. Pentico. Scheduling job shops: Basic methods. In *Heuristic Scheduling Systems. With Applications to Production Systems and Project Management*. 1993.

[62] Zahra Naji Azimi. Hybrid heuristics for Examination Timetabling problem. *Applied Mathematics and Computation*, 163(2):705–733, 2005.

[63] C Nothegger and A Mayer. Solving the post enrolment course timetabling problem by ant colony optimization. *Annals of Operations Research*, 2012.

[64] Oak Ridge National Laboratory. PVM - Parallel Virtual Machine, 2016.

[65] Víctor Peña and Lillo Zumelzu. Estado del Arte del Job Shop Scheduling Problem Introducción Definición del Problema. Technical report, Departamento de Informatica, Universidad Técnica Federico Santa María Valparaiso, 2006.

[66] Nguyen Ba Phuc, Nguyen Tan Tran Minh Khang, and Tran Thi Hue Nuong. A new hybrid GA-Bees algorithm for a real-world university timetabling problem. In *Proceedings - 2011 International Conference on Intelligent Computation and Bio-Medical Instrumentation, ICBMI 2011*, pages 321–326, 2011.

[67] Catalina M. Pineda B. and Lilna M. V., Vargas. *Paralelización del Sistema de Programación Académica*. PhD thesis, Universidad EAFIT, 1998.

[68] Fermin Jesus Pitol-Reyes. *Uso de algoritmos evolutivos para resolver el problema de asignacion de horarios escolares en la facultad de psicologia de la universidad veracruzana*. PhD thesis, Universidad Veracruzana, 2008.

[69] Caballero Fernández Rafael, Molina Luque Julián, Luque Gallego Mariano, Torrico González Angel, and Gómez Nuñez Trinidad. Algoritmos genéticos para la resolución de problemas de Programación por Metas Entera. Aplicación a la Economía de la

Educación. *X Jornadas ASEPUMA / Asociación Española de Profesores Universitarios de Matemáticas para la Economía y la Empresa ASEPUMA*, 2002.

[70] Prapa Rattadilok, Andy Gaw, and Raymond S K Kwan. Distributed choice function hyper-heuristics for timetabling and scheduling. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 3616 LNCS, pages 51–67, 2005.

[71] Piet Rietveld and Martijn Brons. Quality of hub-and-spoke networks; the effects of timetable co-ordination on waiting time and rescheduling time. *Journal of Air Transport Management*, 7:241–249, 2001.

[72] G C W Sabin and G K Winter. The Impact of Automated Timetabling on Universities-A Case Study. *The Journal of the Operational Research Society*, 37(7):689–693, 1986.

[73] Pedro Sánchez Martín and Santiago López. Programación de tareas, un reto diario en la empresa. *Anales de mecánica y electricidad*, pages 24–30, 2005.

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

[75] Martin Schmidt. Solving real-life time-tabling problems. *Foundations of Intelligent Systems*, 1999.

[76] Fawzi Rababa Khaleel Shatnawi, Safwan M Albalooshi. Generating Timetable and Students schedule based on data mining techniques. *International Journal of Engineering Research and Applications*, 2(4):1638–1644, 2012.

[77] A. Jorge Soria-Alcaraz, Martin Carpio, Hector Puga, Jerry Swan, Patricia Melin, Hugo Terashima, and A. Marco Sotelo-Figueroa. *Parallel Meta-heuristic Approaches to the Course Timetabling Problem*, pages 391–417. Springer International Publishing, Cham, 2015.

[78] Daniel Soto, Wilson Soto, and Yoan Pinzón. Una metaheurística híbrida aplicada a un problema de planificación de rutas. *Avances en Sistemas e Informática*, 5(3), 2009.

[79] W Timothy Strayer, Bert J Dempsey, and Alfred C Weaver. *XTP: The Xpress transfer protocol*. Addison Wesley Longman Publishing Co., Inc., 1992.

[80] Arabinda Tripathy. A Lagrangean Relaxation Approach to Course Timetabling. *The Journal of the Operational Research Society*, 31(7):599–603, 1980.

[81] JD Ullman. NP-complete scheduling problems. *Journal of Computer and System sciences*, 393:384–393, 1975.

[82] Cecilia María Velez. Decreto 2566 de 2003, 2003.

[83] Cecilia María Velez. Decreto 1295 de 2010, 2010.

[84] Aitana Vidal. *Algoritmos Heuristicos en Optimizacion*. PhD thesis, Universidad de Santiago de Compostela, 2013.

[85] Aguirre Roldán Viviana and Quintero Diana Carolina. *Modelo de programación de citas para pacientes del proceso integral de rehabilitación (pir) utilizando algoritmos genéticos*. PhD thesis, Universidad de la Sabana, 2006.

[86] Dominic JA Welsh and Martin B Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10(1):85–86, 1967.

[87] George M. White and Simon K S Wong. Interactive timetabling in universities. *Computers and Education*, 12(4):521–529, 1988.

[88] Anthony Wren. Scheduling, timetabling and rostering — A special relationship? In Edmund Burke and Peter Ross, editors, *Practice and theory of automated timetabling*, volume 1153 of *Lecture Notes in Computer Science*, pages 46–75. Springer, 1996.

[89] Liping Wu. The application of Coarse-Grained parallel genetic algorithm with hadoop in university intelligent Course-Timetabling system. *International Journal of Emerging Technologies in Learning*, 10(8):11–15, 2015.

[90] He Yan and Song-nian D. Yu. A multiple-neighborhood-based parallel composite local search algorithm for timetable problem. *Journal of Shanghai University (English Edition)*, 8(3):301–308, 2004.