UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA

FACULTAD DE INGENIERÍA

APLICACIÓN DE ALGORITMOS GENÉTICOS PARA EL PROBLEMA DE ASIGNACIÓN DE HORARIOS EN LA DIVISIÓN DE INGENIERÍAS CIVIL Y GEOMÁTICA

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN INGENIERÍA

INVESTIGACIÓN DE OPERACIONES

P R E S E N T A:

ING. TANYA ITZEL ARTEAGA RICCI

TUTOR:

DRA. IDALIA FLORES DE LA MOTA

Ciudad Universitaria, Mayo 2013

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA
INGENIERÍA EN SISTEMAS – INVESTIGACIÓN DE OPERACIONES

APLICACIÓN DE ALGORITMOS GENÉTICOS PARA EL PROBLEMA DE ASIGNACIÓN DE HORARIOS EN LA DIVISIÓN DE INGENIERÍAS CIVIL Y GEOMÁTICA

T E S I S
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN INGENIERÍA

PRESENTA:
ING. TANYA ITZEL ARTEAGA RICCI

TUTOR PRINCIPAL:
DRA. IDALIA FLORES DE LA MOTA – FACULTAD DE INGENIERÍA

MÉXICO, D. F. MAYO 2013

JURADO ASIGNADO:

Presidente:	Dr. Acosta Flores José Jesús
Secretario:	Dr. Suarez Rocha Javier
Vocal:	Dra. Flores De La Mota Idalia
1 er. Suplente:	Dr. Estrada Medina Juan Manuel
2 d o. Suplente:	M. en C. Espinosa Avila Eduardo

Lugar o lugares donde se realizó la tesis: Ciudad Universitaria, México D.F.

TUTOR DE TESIS:

DRA. FLORES DE LA MOTA IDALIA

FIRMA

Índice general

Índice de figuras

Índice de algoritmos

Índice de tablas

Dedicado a
Mamita Mago
Te amo y siempre en mi corazón estás
Hasta el día en que juntas volvamos a estar

Agradecimientos

Agradezco a Dios por haberme permitido estar aquí, por darme la capacidad necesaria para ser lo que soy, por regalarme su sonrisa y alegrar mi corazón, pero sobre todo por mostrarme el camino y nunca soltarme de la mano.

A la Universidad Nacional Autónoma de México, a la Facultad de Ingeniería y al Posgrado de Ingeniería, por brindarme la oportunidad de estudiar y superarme, por ofrecerme día a día la posibilidad de practicar, hacer y deshacer, de crear, de experimentar, y explotar todo el potencial de ser autónomos y del descubrimiento.

A mi mamita, que siempre me ha apoyado, que ha estado conmigo en las buenas y en las malas, que me brinda su amor incondicional, su apoyo y su fortaleza y que ahora estamos disfrutando juntas de este logro en nuestras vidas.

A toda mi familia, Madito, Esmecita, Lyz, Mimi, Jorge, Ale, Viri, Omar y al conse Carlos, que siempre me acompañan, que están conmigo y que siempre me impulsan a ser mejor como persona y como profesionista, que se quedan conmigo en mis noches de desvelo ofreciéndome amor verdadero.

A Evi y Toyita, a Hiram que me ayudó haciendo cuentas y más cuentas para lograr los resultados esperados de este trabajo, y que aunque todavía no sabe qué va a estudiar, si sabemos que en matemáticas eres muy bueno.

A todos mis revisores que sin su indudable apoyo, ayuda, consejos y sobre todo la paciencia no hubiera podido lograr terminar este proyecto, gracias Ingenieros, Eduardo Espinosa, Alejandro Velázquez; a Erik del Valle, Neithita y Elenita Tai, que me apoyaron mucho en SIG, ¡Que materia tan más complicada!.

A todos mis viejos amigos, a los cuales no terminaría de nombrar y mucho menos de agradecer el apoyo, la dedicación, la fortaleza, el poder escuchar, de estar ahí cuando más los necesito, los quiero mucho y aunque ya no nos vemos tanto como cuando íbamos en la escuela, siempre están en mi mente y en mi corazón.

A todos mis nuevos amigos, en especial, a los chicos de la Unidad de Cómputo, que hicieron posible que este trabajo llegara a su fin, y que con mucho esfuerzo nunca me dejaron caer ni sentirme derrotada, ¿qué sería sin ustedes?, los amo, y con riesgo de omitir a alguien super importante, gracias Tona, Jesús, Lupita, Álvaro, Fer, Dany's Team, Pako, Oly, Fidel, Ulises, Andrés, Ruben, Jair, J.D., Damián, Luis, Cris, Amaury, Anita, Andy, Ernesto, Pepe, Angie, Ezequiel, Alberto, Sergio, David, Ale Guzmán. Gracias Moy, por escucharme, por insistirme en no dejar inconcluso nada, por reír y agitar el puño juntos, porque "el futuro no esta escrito, el futuro lo haces tu mismo", y pronto seremos colegas (Peso!).

Doy gracias especiales a Ulises Acosta, quien desde el día en que nos conocimos sabíamos que lograríamos grandes cosas juntos, y hoy se cumple una más de ellas, gracias por soportarme en mis ratos de crisis y de reír conmigo en los felices, gracias por brindarme tu amor en todo momento y de hacerme sentir que puedo lograr mucho más, gracias por ser parte de este proyecto y de mi vida. Te amo.

A todos ellos y a los que no alcanzó la hoja...

¡GRACIAS TOTALES!

Resumen

Actualmente se tiene un problema de importancia cada inicio de semestre con la asignación de salones para cubrir la demanda académica que se tiene en la División de Ingenierías Civil y Geomática. El método que se utiliza actualmente es poco convencional, lo que lleva a errores de traslape o asignaturas sin salón.

Los problemas de horarios se caracterizan por ser NP-Completos, como se sabe, para estos problemas no existe un algoritmo que resuelva de manera exacta casos grandes en tiempos sensatos, sino pueden tomar meses o inclusos años en encontrar esta solución, por lo que, hay que utilizar técnicas heurísticas para poder encontrar buenas soluciones en un tiempo razonable de cómputo. Este trabajo proporciona una herramienta efectiva para la solución de problemas de horarios bajo un esquema de poblaciones muy grandes.

PALABRAS CLAVE: Problemas de horarios, heurística, coloración de grafos, coloración robusta, algoritmos genéticos.

Abstract

At the beginning of each academic semester the Civil and Geomathic Engineering Division has an essential problem with allocation of classrooms in order to satisfy the academic demand. The method currently used is outdated, which leads to errors of overlapping or asignature without asignated classroom.

The scheduling problems are characterized as NP-complete, so it is known that for these class of problems there are not exact algorithm to solve huge cases in acceptable time, it can last lot of time even years to find the optimal solution, that is why the use alternative technics currently used to find good solutions or even optimal in a reasonable computation time. This work provides an effective tool to solve scheduling problems for big size problems.

KEYWORDS: scheduling problems, heuristics, graph coloring, coloring robust, genetic algorithms.

Introducción

Elaborar una asignación óptima de horarios, es un problema por el que atraviesa con una periodicidad específica cualquier institución educativa, sobre todo cuando la población de estudiantes y planta académica es extensa; considerando que se puede llegar a complicar exponencialmente cuando esta población crece de manera importante con relación a su infraestructura.

Desde la perspectiva de la Investigación de Operaciones, este tipo de problemas se enmarcan dentro del área conocida como programación de horarios (*Timetabling*), se caracteriza por ser un problema NP-Completo [20], y resulta un problema de optimización combinatoria bien estudiado y con una amplia aplicación. Aunado a esto, como ya se mencionó, la creciente población de estudiantes y los cambios recientes a los programas de estudio incluyendo más asignaturas y mejorando el plan de estudios, han resultado en un incremento a la complejidad de poder encontrar una solución factible. Los problemas de esta área consisten en la asignación de ciertos eventos a distintos bloques de horarios respetando una serie de requerimientos y condiciones impuestas por el personal académico y administrativo de la institución.

Estas condiciones o requerimientos pueden ser del tipo:

- Solo se dispone de una persona para realizar la asignación.
- Dependiendo del semestre cambia el número disponible de salones.
- Se tienen cerca de 200 grupos por semestre aproximadamente.
- Se tienen cerca de 180 profesores aproximadamente.
- Se tiene prioridad en profesores de asignatura sobre profesores de carrera.
- Los salones tienen un límite de alumnos (capacidad).

- En las salas de cómputo no deben encontrarse más de 20 alumnos, por lo que los grupos que cuenten con más alumnos deberán ser reasignados y en su caso abrir un grupo más.
- Las clases comienzan a las 07:00 hrs y terminan a las 22:00 hrs.

El método que se utiliza es poco convencional, ya que se hace a mano y colocando cada una de las clases en un horario según la consideración de las personas que lo llevan a cabo, lo que lleva a errores de traslape o asignaturas sin salón. Se tienen que considerar diversos puntos de importancia como ya se mencionó anteriormente.

De lo anterior, se deriva la importancia de tener un control bien planificado sobre la forma en que se llevará a cabo la utilización de recursos materiales y humanos, no basta con tener excelentes profesores y alumnos dedicados, también se debe ofrecer la infraestructura en donde la impartición de las asignaturas tenga a bien definirse.

Dentro de este entorno, se considerará el poder programar de manera eficiente, efectiva y eficaz, los horarios de clases en sus respectivos salones, tomando en cuenta la disponibilidad de estos así como, el número de profesores que pueden impartir tales asignaturas, y la capacidad de estudiantes que nos brinde el espacio de las aulas.

Problemas de este tipo, se pueden modelar como problemas de coloración [13] dependiendo del tipo de restricciones que se consideren, se puede utilizar una generalización de dicho problema, una de estas es el problema de coloración robusta generalizada, [35], y entre los métodos usados para encontrar buenas soluciones están el recocido simulado [30], búsqueda tabú [4], algoritmos genéticos, sistemas expertos, algoritmos voraces y modelos empíricos. Se debe mencionar que el problema está acotado solamente a considerar la División de Ingenierías Civil y Geomática de la Facultad de Ingeniería de la Universidad Nacional Autónoma de México, la cual tiene en promedio 70 asignaturas y cerca de 180 profesores para todas ellas; no se debe olvidar por supuesto, que aunque el tiempo promedio de esta asignación se disminuirá drásticamente, no está libre de errores, sin embargo, una buena programación generará una serie de beneficios para los principales actores. Ya que se puede ofrecer una mejor atención, una rápida resolución y por supuesto un inicio de clases sin mayores percances.

Objetivo general

Este trabajo tiene como objetivo desarrollar un algoritmo para la asignación de horarios en la División de Ingenierias Civil y Geomática, de tal forma que se minimicen los empalmes en los salones, la ausencia de disponibilidad para la impartición de alguna asignatura y evitar la falta de profesores en las asignaturas existentes.

Objetivos Específicos

Como objetivos específicos están:

- Dar un panorama sobre la programación de horarios y como se han atacado este tipo de problemas, comprender las características y propiedades referentes a técnicas heurísticas y conocer sus aplicaciones.
- Dar a conocer los antecedentes con los que cuenta la División de Ingenierías Civil y Geomática y las herramientas que actualmente usan, describir la problemática y la necesidad de implementar un algoritmo que satisfaga esas vulnerabilidades.
- Definir las estructuras de información y operadores que debe usar un algoritmo genético que pueda aplicarse al problema de asignación de horarios.
- Especificar, diseñar, implementar y evaluar un modelo matemático que incorpore a los algoritmos genéticos para explorar el espacio de soluciones determinando la calidad de esas soluciones obtenidas y utilizar medidas de evaluación apropiadas que permitan compararlas con resultados previos.

Estructura general del trabajo

El presente trabajo de Tesis, se encuentra estructurado de la siguiente manera.

En el primer capítulo se muestra el análisis que se hizo sobre la problemática que presenta la programación de horarios, así como las herramientas que se utilizarán para resolver dicha problemática, teniendo como columna vertebral a los algoritmos genéticos, proponiendo un modelo de programación matemática con restricciones adicionales.

En el segundo capítulo se aborda el marco teórico, estado del arte y algunos conceptos téoricos que se utilizarán más adelante referentes a la problemática a tratar.

En el tercer capítulo se muestra las bases teóricas de los algoritmos genéticos, asi como sus operadores más representativos y la forma de implementación que se considerará para el desarrollo del modelo matemático.

Para el cuarto capítulo, se muestra el desarrollo del modelo matemático y su planteamiento, el cual tiene fundamentaciones sobre otros trabajos [6, 35, 18] con esto se logra desarrollar un planteamiento más claro y sobre todo una aplicación real. Después vendrá la implementación y se estudiará su comportamiento, como ya habíamos visto, sobre una situación real y actual.

En el quinto capítulo se muestra la implementación y su validación, en donde se muestran los resultados de las pruebas que se hicieron antes de la implementación definitiva, con la finalidad de adecuar o mejorar, si fuera el caso, a la función objetivo y/o a las restricciones del modelo.

Finalmente se muestran las conclusiones del trabajo y se se da pie o la base a futuros desarrollos sobre esta misma línea que pueden ser implementados para problemas con menos restricciones y variables de entrada, o con muchas variables de entrada y restricciones como lo es en este caso.

Al final de este documento se cuenta con varios apéndices en donde se puede observar segmentos del código en las partes más relevantes del algoritmo propuesto.

Capítulo 1

Análisis de la Problemática de la Programación de Horarios

1.1. Antecedentes

Dadas las condiciones de horarios en la División de Ingenierías Civil y Geomática, en donde las asignaturas no tienen un patrón claro, es decir, hay asignaturas que se imparten una o dos veces a la semana con el mismo número de horas por semana, y otras que se imparten dos o tres veces a la semana con igual número de horas (Tabla 1.1), esto muchas veces depende del tipo de asignatura y también del profesor que la impartirá, para efectos prácticos de este trabajo se realizó una estadística que demuestra cual es el horario dominante para poder realizar el modelo de programación.[1]

[1] Vease Anexo A para tabla completa

Asignaturas que se imparten en la División de Ingenierías Civil y Geomática

	No. de horas Diarias				Semanal
	1.5	2.25	3	4.5	
Administración en Ingeniería	4		1		3
Construcción I				1	4.5
Construcción II	1				3
Construcción III			1		3
Movimiento de Tierras	4		1		3
Presupuestación de Obras	4	4		1	4.5
Programación y Construcción de Estructuras	3	5			4.5
Análisis Estructural	8	1			4.5
Diseño Estructural	6	1			4.5
Estatica Estructural	8	3			4.5
Estructuras Isostaticas	1				4.5
Mecánica de Materiales I	9				4.5
Mecánica de Materiales II	7	1			4.5
Proy. Estruct. P/Edific. Conc y Mamp.	1	1			4.5
Proyecto de Estructuras Metálicas	1	1			4.5
Temas Especiales de Ing. Civil I	1	2			
Temas Especiales de Ing. Civil II	?				
Cimentaciones					
Comportamiento de Suelos					
Fundamentos de mecánica					
Geología					

Tabla 1.1: Duración de asignatura a la semana

Con lo que también se concluye que existen cerca de 200 grupos que se imparten solamente 1.5 horas por día 2 o 3 veces por semana y pocos grupos con 2.25 horas 2 veces a la semana, que en realidad resulta de 4.5 horas a la semana, por lo que se tomará que por estandar el horario de clase sea de 1.5 horas cada vez de 2 a 3 veces a la semana, lo anterior con base al resultado que se dió de un análisis de los últimos semestres, no olvidando que durante este tiempo se vinieron dando una serie de cambios a los planes de estudio y es por esto que se tuvo que hacer un promedio de horarios con asignaturas.

En la figura 1.1 se puede observar la gráfica del número de horas que se tiene, esto sirve para poder determinar con cuantos intervalos de 1.5 horas se puede contar para realizar la asignación.

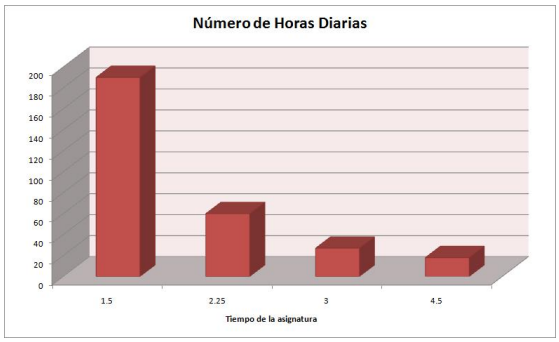

Figura 1.1: Duración de las asignaturas por día

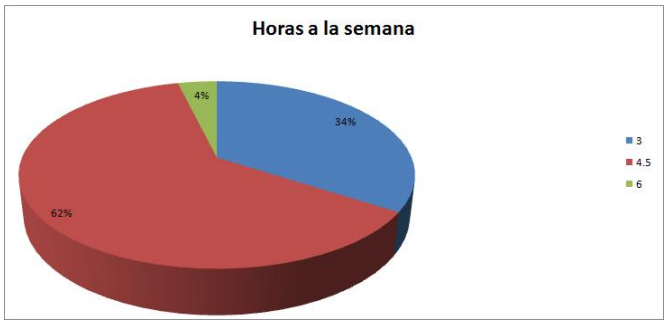

Figura 1.2: Duración de las asignaturas por semana

En la figura 1.3 y la figura 1.4 se puede apreciar los horarios en los que se puede tener asignación de clases, dando como resultado 45 intervalos de tiempo, uno para cada salón.

	LUNES	MARTES	MIÉRCOLES	JUEVES	VIERNES
7:00 – 8:30	1	10	19	28	37
8:30 – 10:00	2	11	20	29	38
10:00 – 11:30	3	12	21	30	39
11:30 – 13:00	4	13	22	31	40
13:00 – 14:30	5	14	23	32	41

Figura 1.3: Primer conjunto de horarios disponibles

	LUNES	MARTES	MIÉRCOLES	JUEVES	VIERNES
16:00 – 17:30	6	15	24	33	42
17:30 – 19:00	7	16	25	34	43
19:00 – 20:30	8	17	26	35	44
20:30 – 22:00	9	18	27	36	45

Figura 1.4: Segundo conjunto de horarios disponibles

También se debe tener en cuenta el número de materias que se imparten por semestre (Figura 1.5), ya que una de las restricciones es que los alumnos que van de manera regular puedan inscribir todas las materias que les corresponde de acuerdo al semestre que se esta cursando, esto con el fin de que no haya empalmes.

Semestre	Bloque	Asignaturas	Asignaturas	Horas
Primero	I		TI	4.5
Segundo	II	GTIA	TII	4.5
Tercero	III	EE, PO	FI	4.5
Cuarto	IV	MMI, FMMC, PCE	FII, SC	4.5
Quinto	V	MMII, HB, TGS, IAMRM	TIII, FG, TEI,MM	4.5
Sexto	VI	AE, GGIA, HMT, IS	GDIA, CGA, TEII, AP, HAG, GG	4.5
Séptimo	VII	DE, CS, HC, MT, P	PRI, SIGI, GPS	4.5
Octavo	VIII	MS, HDIA, ST, AGPA, PEM, PEECM, EPV	PRII, SIGII, PG, EG, GHM, HDFIA, LT, PGM, TYM	4.5
Noveno	IX	EP, AI, IP, HU, CMT, OH, TACH, TAR		4.5

Figura 1.5: Concentrado de asignaturas por semestre

En las tablas 1.2 y 1.3 se muestran las materias que se imparten, el semestre y las horas de cada una.

Clave	Nombre	Semestre	Semanal
AGPA	Abastecimiento de Agua Potable y Alcantarillado	8	4.5
AI	Administración en Ingeniería	9	3
AE	Análisis Estructural	6	4.5
CS	Comportamiento de Suelos	7	4.5
DE	Diseño Estructural	7	4.5
EE	Estática Estructural	3	4.5
EP	Evaluación de Proyectos	3	3
FMMC	Fundamentos de Mecánica del Medio Continuo	4	3
GGIA	Geología	6	4.5
GTIA	Geomática	2	4.5
HB	Hidráulica Básica	5	4.5
HC	Hidráulica de Canales	7	4.5
HMT	Hidráulica de Máquinas y Transitorios	6	4.5
HDIA	Hidrología	8	4.5
IAMRM	Impacto Ambiental y Manejo de Residuos Municipales	5	4.5
IS	Ingeniería de Sistemas	6	4.5
IP	Integración de Proyectos	9	3
MMI	Mecánica de Materiales I	4	4.5
MMII	Mecánica de Materiales II	5	4.5
MS	Mecánica de Suelos	8	4.5
MT	Movimiento de Tierras	7	3
P	Planeación	7	3
PO	Presupuestación de Obras	3	4.5
PCE	Programación y Construcción de Estructuras	4	4.5
ST	Sistemas de Transporte	8	4.5
TGS	Teoría General de Sistemas	5	3
CMT	Cimentaciones	9	4.5
EPV	Estructuras de Pavimentos	8	4.5
HU	Hidráulica Urbana	9	4.5
OH	Obras Hidráulicas	9	4.5
PEM	Proyecto de Estructuras Metálicas	8	4.5
PEECM	Proyecto Estructural para Edificiaciones de Concreto y Mampostería	8	4.5
TACH	Tratamiento de Agua para Consumo Humano	9	4.5
TAR	Tratamiento de Agua Residual	9	4.5

Tabla 1.2: Nomenclatura de asignaturas para Ing. Civil

Clave	Nombre	Semestre	Horas
AP	Administración de Proyectos	6	3
CGA	Cartografía	6	3
FI	Fotogrametría I	3	4.5
FII	Fotogrametría II	4	3
FG	Fundamentos de Geodesia	5	3
GDIA	Geodesia	6	4.5
GG	Geología y Geomorfología	6	3
HAG	Hidrología Aplicada a la Geomática	6	4.5
MM	Modelación Matemática	5	3
PRI	Percepción Remota I	7	3
PRII	Percepción Remota II	8	3
PG	Proyecto Geomático	8	3
GPS	Sistema de Posicionamiento Global	7	3
SC	Sisemas de Coordenadas	4	3
SIGI	Sistemas de Información Geográfica I	7	3
SIGII	Sistemas de Información Geográfica II	8	3
TEI	Teoría de los Errores I	5	3
TEII	Teoría de los Errores II	6	3
TI	Topografia I	1	4.5
TII	Topografia II	2	4.5
TIII	Topografía III	5	4.5
APC	Automatización de Procesos Cartográficos	8	4.5
CRO	Catastro	8	4.5
EG	Exploración Geofísica	8	4.5
GHM	Geología Histórica y de México	8	4.5
HDFIA	Hidrografía	8	4.5
LT	Legislación Topográfica	8	3
PGM	Prospección Gavimétrica y Magnetométrica	8	4.5
TYM	Topografía de Yacimientos Minerales	8	3

Tabla 1.3: Nomenclatura de asignaturas para Ing. Geomática

1.2. Teoría de Gráficas

La teoría de gráficas proporciona algoritmos eficientes para resolver problemas en distintos campos de la ciencia. Se definen las gráficas de la siguiente manera:

Una gráfica G está formada por dos conjuntos, V el conjunto de vértices y A es el conjunto de las aristas.

Orden de una gráfica: es el número de vértices o nodos que tiene.

Tamaño de la gráfica: es el número de aristas o arcos que tiene.

Incidencia: Dado un vértice i, su grado de incidencia es el número de extremos de i.

Gráfica completa: Si todo par de vértices (i,j) se encuentra unido por una arista o un arco como mínimo.

Gráfica incompleta: si no se cumple la condición anterior.

Gráfica simétrica: si para todo par de (i,j) existe el par (j,i).

1.2.1. El Problema de Coloración de Gráficas

El problema de coloración de gráficas consiste en que dada una gráfica $G = (V, A)$ y un conjunto $c = \{1, 2, ..., k\}$ de colores hay que *pintar* los vértices con el menor número k de colores, de tal forma que cualquier par de vértices adyacentes no tengan el mismo color [35, 21, 10].

En la asignación de horarios, el problema de coloración es usado para representar cada clase por un vértice y las aristas son creadas entre cada par de vértices que correspondan a clases que no pueden ser programadas a la misma hora. La no disponibilidad y las preasignaciones son manejadas imponiendo alguna restricción externa sobre la viabilidad de coloreo de un vértice específico. La coloración de la gráfica resultante puede fácilmente volverse una asignación de horarios práctica asociando cada horario con un color.

Los primeros reportes que se tienen sobre que el problema de asignación de exámenes o clases a periodos, resultaba equivalente a asignar colores a los vértices de una gráfica, fué a mediados de los años sesenta. Cada vértice está entonces *coloreado* de tal forma que ningún otro al que esté conectado tenga ese mismo *color*. Entonces, si cada color representa un período diferente, se puede asegurar que no se han programado clases que deben impartirse en el mismo periodo [9, 35].

El término técnico para el mínimo número de colores necesarios para una apropiada coloración de una gráfica es *número cromático* denotado por $\gamma(G)$. El cual es equivalente al mínimo número de colores que se necesitan para colorear la programación de un conjunto de clases evitando cualquier conflicto. Si el *número cromático* es menor que o igual al número de periodos disponibles para las clases entonces se

sabe que se ha encontrado una solución sin conflictos, aunque esto no significa que sea una solución viable ni útil, como se puede apreciar en la figura 1.6. *Fuente: http://mathworld.wolfram.com/ChromaticNumber.html.*

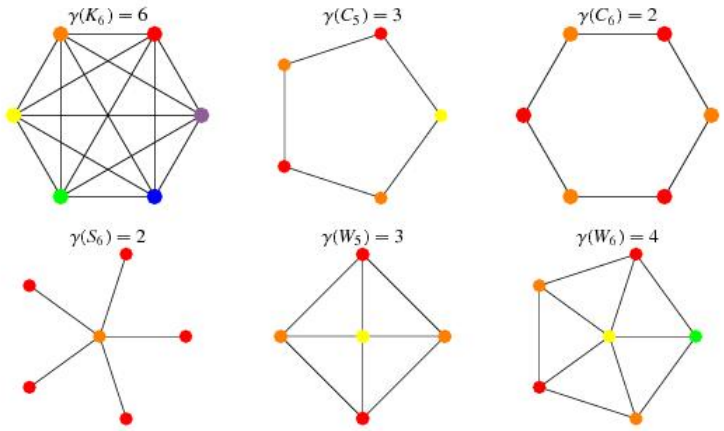

Figura 1.6: Número Cromático.

1.2.2. Problema de Coloración Robusta

En [35] se define el problema de coloración robusta considerando una gráfica $G = (V, A)$ y su gráfica complementaria $G = (\bar{V}, \bar{A})$ donde $\bar{A}$ es el conjunto de aristas que no están en G. Dada una función de peso definida sobre las aristas de $\bar{G}$, se dice que la rigidez de una *k-coloración* válida de G es la suma de los pesos de las aristas de $\bar{G}$ que unen vértices del mismo color. Con base en la anterior definición, se plantea el ***Problema de Coloración Robusta*** al buscar la *k-coloración* válida de rigidez mínima. Una coloración válida identifica una asignación de recursos compatibles. Distintas penalizaciones de las aristas complementarias permiten obtener coloraciones válidas con diferentes propiedades. En particular, se puede considerar el caso en que se añaden nuevas aristas a la gráfica y algunas coloraciones continuarán siendo válidas, otras no, la coloración que sigue siendo válida se llama coloración robusta. Con el menor grado de rigidez de una coloración se obtiene la coloración más robusta y es la que interesa encontrar.

Existen problemas de programación de actividades o programación de horarios, como lo es el de este trabajo que se pueden plantear como problemas de coloración de los vértices de una gráfica, frecuentemente tratado de minimizar el número de colores a utilizar.

En [35] se demuestra que el problema de coloración robusta es una generalización del problema de coloración mínima (PCM), lo que permite modelar problemas con restricciones adicionales a las que tienen los problemas modelados por el PCM. Aquí el objetivo no es minimizar el número de colores utilizados en la coloración de los vértices, lo que interesa es que una solución del problema sea lo más estable posible, en el sentido de que al añadir aristas a la gráfica la coloración continúe siendo válida.

Es importante recordar que la complejidad del problema que se quiere modelar en este trabajo, no solo es representada por la complejidad computacional del algoritmo en si, sino por todas las consideraciones que se deben tomar en cuenta ya que se trata de un problema que involucra al carácter humano, en donde, algún profesor en particular y debido a su trayectoria dentro de esta Facultad, tiene preferencia sobre algunos horarios y salones para impartir su clase, es por esto que no solo se puede restringir a un PCM. En estos problemas es común que el objetivo sea utilizar el mínimo número de colores, de tal manera que a cada par de elementos a programar que no pueden compartir el mismo recurso se les asigne un color diferente. Cada clase de color obtendrá los elementos entre los que no hay conflicto y que pueden compartir el recurso en cuestión.

En [35, 10] se muestra que los vértices representan los elementos a programar y cada arista identifica la incompatibilidad entre los elementos correspondientes. Esta incompatibilidad representa el conflicto entre recursos cuando no puedan ser compartidos.

En [35] se presentan varios ejemplos de la utilización de esta técnica, en este caso se tomará unicamente al problema de programación de horarios.

1.3. Problema de Coloración Robusta Generalizada

El problema de coloración robusta generalizado (PCRG) [36, 41, 1], definido en [35], considera una gráfica G con un conjunto de vértices V, un conjunto de aristas A y un conjunto de aristas complementarias $\bar{A}$. En el PCRG ambos conjuntos de aristas existen en la gráfica e introduce la definición de distancia entre los colores que se define para cada par de vértices unido por una arista complementaria.

Puede ocurrir además que la programacion de horarios este limitada por restricciones temporales del tipo *«dos eventos no pueden ser programados en el mismo día»* o *«debe haber al menos dos días entre dos clases de la misma asignatura y grupo»* también con

restricciones de espacio, como no pueden programarse más de k clases en cada hora por disponer de un número limitado de salones. En estos casos el problema de coloración robusta no resuelve el problema, no obstante, un nuevo problema de coloración de gráficas permitiría abordarlos, en que se relaja el concepto de incompatibilidad de una coloración respecto al problema de coloración robusta.

Con esta idea, los 45 colores (Figura 1.3 y Figura 1.4) que se tienen disponibles repartidos en los cinco días de la semana se asocian ordenadamente a los colores: el color 1 para la primera hora del lunes, el color 2 para la segunda hora y así sucesivamente. Por otra parte, y aprovechando la flexibilidad que incorpora en los problemas de coloración el grado de rigidez, se relaja la gráfica al suprimir las aristas que unen las clases de una misma asignatura y se mantienen las aristas que unen las clases de un mismo semestre pero de distinta asignatura. Obviamente, y para garantizar que no sean coloreadas igualmente dos clases de la misma asignatura, deben ser penalizadas fuertemente este tipo de aristas de la gráfica complementaria.

Capítulo 2

Marco Teórico

Un elemento principal en la investigación de operaciones es el modelado matemático. Aunque la solución del modelo matemático establece una base para tomar una decisión, se deben tener en cuenta factores intangibles o no cuantificables, por ejemplo, el comportamiento humano, para poder llegar a una decisión final. Dentro de la búsqueda por obtener el modelo matemático que pueda representar al problema que se está atacando, se puede considerar dos tipos de métodos: los exactos y los heurísticos, sin embargo, se debe tener conciencia de cual es la realidad del problema y su solución, en la elección de alguno de los dos.

2.1. Métodos Exactos

La mayor parte de los problemas pueden ser clasificados como prescriptivos o de optimización. Un modelo de este tipo «dicta» el comportamiento para una organización que le permitirá a ésta alcanzar mejor su meta. Entre los elementos de un modelo se encuentra:

- *Variable de decisión*
- *Función objetivo*
- *Restricciones*

Es decir, un modelo de optimización trata de encontrar valores, entre el conjunto de todos los valores para las variables de decisión, que optimicen (maximicen o minimicen) una función objetivo que satisfaga las restricciones dadas.

Las variables de decisión están bajo nuestro control e influyen en el desempeño del sistema. Las restricciones son, igualdades ($=$) o desigualdades ($<$, $>$, $\leq$, $\geq$), y sólo son posibles ciertos valores de las variables de decisión.

Algunos métodos de solución para estos problemas son: planos de corte, ramificación y acotamiento, teoría de grupos, programación dinámica, entre otros.

Estos métodos encuentran siempre la mejor solución, es decir el óptimo en el problema (ya sea el máximo o el mínimo). Sin embargo, el carácter no polinomial de una buena parte de los problemas de optimización combinatoria, hace imposible o realmente difícil utilizar un método exacto, ya que la computadora se tarda días e incluso años en encontrar la mejor solución. En estos casos se debe buscar otra alternativa para encontrar una solución al problema y se hace necesario contar con buenos métodos heurísticos.

2.2. Métodos Heurísticos

La palabra heurística procede del griego $\varepsilon\upsilon\rho\iota\sigma\kappa\varepsilon\iota\nu$, que significa «*encontrar, descubrir*», en Investigación de Operaciones, heurística se describe mejor como método de búsqueda. Una heurística es un procedimiento de resolución de problemas de optimización bien definidos para el que se tiene un alto grado de confianza en que se encuentren soluciones de alta calidad con un costo computacional razonable (tiempo polinomial), aunque no garantice la optimalidad y en algunos casos ni la factibilidad.

Los procedimientos metaheurísticos son una clase de métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria, en los que los heurísticos clásicos no son ni efectivos ni eficientes. Los metaheurísticos proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de: inteligencia artificial, evolución biológica y mecanismos estadísticos.

Actualmente, existe una cantidad muy importante de trabajos científicos publicados que abordan problemas de optimización a través de las metaheurísticas, investigaciones sobre nuevas heurísticas o extensiones de las metaheurísticas.

Estos métodos son basados en la experiencia, como su nombre lo dice. Con el transcurso del tiempo, se han encontrado varias metaheurísticas diferentes que dan muy buenos resultados dependiendo del tipo de problema que se trate. Estos métodos, ya que no son exactos, no prometen encontrar el óptimo, pero si una muy buena solución.

Dentro de esta rama se pueden encontrar los siguientes métodos: constructivos, de descomposición, de reducción, manipulación del modelo, búsqueda por entornos, en este último, tiene como objetivo perfeccionar una solución existente, mediante una búsqueda local en una vecindad bien definida del espacio de soluciones; entre estos métodos encontramos, recocido simulado, búsqueda tabú, algoritmos genéticos, redes neuronales y GRASP.

2.2.1. Métodos Heurísticos en la Programación de Horarios

Para resolver el problema de la programación de horarios se han diseñado diversos algoritmos y se han aplicado diversas estrategias de solución.

Las estrategias basadas en heurísticas directas involucran el poner, en una programación de horarios, todas las sesiones que sean posible hasta que se originen conflictos. En este punto, las sesiones o clases son intercambiadas en un intento de permitir la programación de otra clase. Generalmente, la mejor estrategia es programar las sesiones en orden, a partir de aquella que esté mas ajustada [18].

El método que se presenta en este trabajo es fundamentalmente una ayuda para la técnica actual de la programación de horarios. Los métodos heurísticos han funcionado muy bien en situaciones similares por lo que también se basará en problemas del mismo tipo ya estudiados [39, 12, 6, 35] y se acoplarán a nuestra situación en particular. En primer lugar se modela el problema y las situaciones que puedan surgir a lo largo de la resolución usando la heurística específica que fue elegida. Después, se perfecciona la base y se descartan los casos que no resultan ser útiles para resolver nuevos problemas, es decir, después de considerar un caso en particular se debe poder generalizarlo para cualquier especificación que se pueda dar posteriormente.

La gran variedad de problemas que pueden ser incluidos en el campo de la programación y asignación de horarios, el hecho de que los métodos de educación estan cambiando, así como las facilidades computacionales que nos brindan las tecnologías de información actuales y la disponibilidad que se tiene en las instituciones educativas, significa

una gran oportunidad para el campo de la Investigación de Operaciones mediante la utilización de métodos heurísticos.

2.3. Teoría de la Programación de Horarios

La palabra *scheduling* (programación) es un término dentro de nuestro vocabulario, a pesar de esto podría no tener una buena definición sobre este. De hecho, la programación no es un concepto común en nuestra vida diaria, en lugar de eso es la calendarización. Una programación de horarios es un plan tangible o documento tal como un horario de autobús o un horario de clases [7], la programación usualmente dice cuándo se supone que deben pasar las cosas, y se muestra un plan de tiempo con ciertas actividades y responde a la pregunta de ¿Si todo va bien?, ¿Cuándo sucederá un evento en particular?.

Suponiendo que se está interesado en la hora que será servida la cena o la hora en que va a salir un autobús, en estas instancias, la referencia del evento es la terminación de cierta actividad particular, tales como preparar la cena, o el inicio de alguna actividad como la partida de algún transporte público. Las respuestas a la pregunta «*¿dónde?*», generalmente vienen con la información acerca del tiempo, la cena está programada para servirse a las 6:00 pm, el autobús esta programado para partir a las 8:00 am. Sin embargo, una respuesta podría ser igualmente útil en términos de secuencia en lugar de tiempo: es decir, la cena será servida en cuanto el plato fuerte salga del horno, o el autobús saldrá de la estación después de que se haya terminado la limpieza y el mantenimiento. Así, la pregunta «*¿cuándo*?» puede ser contestada por tiempo o por información de la secuencia obtenidos a partir de la programación.

Si se tiene en cuenta que algunos eventos son impredecibles, a continuación, los cambios pueden ocurrir en un horario o en una programación de actividades. Incluso entonces, el programa es útil: al permitir que los pasajeros sepan cuándo el autobús se debe dejar, le ayuda a planificar sus propios horarios, aún cuando éste tenga cierta incertidumbre inherente.

Por asociación, cuando se piensa en programación se piensa en generar un horario, en él se puedan abrir «casillas» en donde se colocan las actividades, sin embargo y no comúnmente, no se detiene a considerar los detalles de los que podría ser este proceso. De hecho, aunque se piensa en un calendario como algo tangible, el proceso de programación parece bastante intangible, hasta que se considera con cierta profundidad. A menudo

este enfoque se puede dividir en dos fases: *la secuenciación y programación*. En la primera fase, se debe decidir la secuencia o el cómo se seleccionará la siguiente tarea o actividad. En la segunda fase, se planea la hora de inicio, y en dado caso, el tiempo de terminación de cada tarea. No se debe dejar de lado la importancia de contar con un tiempo de seguridad que brindará una holgura, esto debido a la incertidumbre que se puede encontrar, cabe mencionar que este último tiempo se considera en la segunda fase.

Los problemas de programación en la industria tienen un conjunto de tareas y un conjunto de recursos disponibles para llevar a cabo esas actividades. Dados esas tareas y recursos, junto con alguna información acerca de las incertidumbres, el problema general es determinar el tiempo de las tareas sin dejar de reconocer la capacidad de los recursos.

En el proceso de la programación de actividades se necesita saber el tipo y la cantidad de cada recurso para determinar el tiempo factible para realizar cada tarea o actividad. Estas tareas se deben describir en términos de los requerimientos de cada recurso: la duración, el tiempo más temprano en el que puede comenzar la tarea y el tiempo en el que debe ser terminada (o fecha de entrega). En general, el tiempo de duración de una actividad es incierto, sin embargo en muchos casos se omite al principio este hecho para plantear el problema.

En general se puede decir que un problema de programación de horarios está definido por la información acerca de los recursos y de las tareas o actividades. Resolver estos problemas puede ser altamente complejo, es por eso que son de gran ayuda las soluciones de problemas formales. Los modelos formales ayudan a entender el problema para encontrar, posteriormente, una buena solución.

2.4. Problema de Horarios

El problema de crear horarios válidos consiste en la programación de grupos, profesores y salones dentro de un determinado periodo, de tal manera que ningún profesor, grupo o aula escolar se utilice más de una vez por periodo, entendiendo por periodo, al tiempo en el que se estará ocupando un salón de clases, con la impartición de alguna asignatura, evidentemente también un académico estará en modo ocupado. Por ejemplo, si un grupo debe impartirse dos veces a la semana, entonces debe ser colocado en dos períodos

diferentes para evitar una colisión. Un grupo se compone por un número de estudiantes. Se supone que la asignación de los estudiantes en los grupos es fijo, y que los grupos son disjuntos, es decir, *no tiene estudiantes en común.*

Bajo este esquema [3, 26], una correcta programación de horarios es aquella en la que un grupo puede ser programado al mismo tiempo que cualquier otro grupo. En cada período se enseña una materia. Una combinación particular de un profesor, una asignatura, un salón de clases se denomina tupla (*tuple*) [3]. Una tupla puede ser calendarizada más de una vez por semana. Así, el problema de horarios se puede expresar como la programación de una serie de tuplas tal que, un profesor impartiendo la clase o el salón no aparece más de una vez por periodo.

La tarea de combinar los grupos, los profesores y el salón de clases dentro de una tupla es manejada como operaciones separadas. Las tuplas son formadas bajo la primicia de saber los requisitos de las diversas clases y profesores, así como la información sobre la disponibilidad de los salones. Es conveniente dividir el problema en esta forma ya que reduce la complejidad de la tarea de programación.

2.5. Optimización Combinatoria

La optimización combinatoria es una parte de la programación matemática que estudia problemas del tipo:

$$min\{f(x) : x \epsilon S\}$$

siendo $|S| < \infty$, es decir, la optimización combinatoria trata de desarrollar algoritmos para afrontar problemas de optimización caracterizados por tener un número finito de solución factibles. Aunque por definición puede parecer que la «simple» enumeración de las soluciones en S es suficiente para resolverlos, esta idea no es realizable en la práctica, ya que el número $/S/$ puede ser exageradamente enorme. Por otra parte, particularidades del conjunto S permiten en ocasiones diseñar algoritmos efectivos que permiten determinar (con tiempos de cálculo razonables) alguna solución al problema.

2.6. Complejidad Computacional

La complejidad computacional [17, 32] es la rama de la teoría de la computación que estudia, de manera teórica, la complejidad inherente de problemas de importancia práctica y/o teórica a la resolución de un problema computable. Los recursos comúnmente estudiados son el tiempo (mediante una aproximación al número y tipo de pasos de ejecución de un algoritmo para resolver un problema) y el espacio (mediante una aproximación a la cantidad de memoria utilizada para resolver un problema).

La eficiencia de los algoritmos deja en claro que depende en gran medida de los avances en cuanto a tecnologias se refiere, sin embargo, esto no marca un hecho con el que se puedan distinguir lo mejor de un algoritmo o de otro, como ya se dijo, puede haber casos particulares que se comportan mucho peor que el promedio y nadie puede tener la confianza, en su suerte, al intentar resolverlo. La cantidad de operaciones elementales depende del tamaño de los datos del problema a estudiar. El poder distinguir alguna clase de patron en 10 números es mucho muy diferente a querer buscar en miles de millones de números, de tal forma, se expresa en número de operaciones elementales como una función de algún o algunos números característicos suficientes, para calcular el volumen del trabajo realizado.

Se pueden estudiar igualmente otros parámetros, tales como el número de procesadores necesarios para resolver el problema en paralelo. La teoría de la complejidad difiere de la teoría de la computabilidad en que ésta se ocupa de la factibilidad de expresar problemas como algoritmos efectivos sin tomar en cuenta los recursos necesarios para ello. Los problemas que tienen una solución con orden de complejidad lineal son los problemas que se resuelven en un tiempo que se relaciona linealmente con el tamaño de la entrada.

Se sabe que un algoritmo que resuelve un problema en tiempo exponencial es de utilidad limitada. El tiempo de ejecución de este tipo de algoritmos crece exponencialmente con el tamaño de la entrada, de ahí que ocupa demasiado tiempo en resolver inclusive instancias pequeñas. Es por esto que siempre o en la mayoría de los casos se busca poder reducir la complejidad de dichos problemas a un tiempo polinomial, estos tipos de problemas son en realidad más eficientes y se utilizan todo el tiempo. La clase de problemas que enfrentan los algoritmos polinomiales se denomina tipo P.

Sin embargo, no todo se puede expresar en forma polinomial, los matemáticos han conseguido demostrar que existen problemas tan difíciles que no existe manera de

resolverlos en tiempo polinomial mediante una computadora. Existen algunos problemas que a simple vista se ven muy sencillos como son las torres de Hanoi y muchos otros no tienen solución eficiente, existe otra clase importante de problemas para los cuales no se conocen soluciones eficientes pero no se ha logrado probar que no la tengan.

La programación de horarios y asignación de salones es una larga y compleja tarea, se cuentan con diferentes departamentos y laboratorios, además los problemas de esta área consisten en la asignación de ciertos eventos a distintos bloques de horarios respetando una serie de requerimientos y condiciones impuestos por el personal administrativo de esta institución. La programación de horarios pertenece a la clase de problemas llamados *NP-Completos* [23], es decir, que es necesario utilizar algoritmos aproximados para encontrar buenas soluciones en un tiempo razonable de cómputo y el tamaño del espacio de soluciones crece de manera importante con respecto al número de variables.

Por lo tanto, es importante desarrollar métodos eficientes de búsqueda para llegar a una solución óptima o casi óptima que cae dentro de los límites establecidos [25].

Las facilidades que ofrecen las tecnologías de información, en particular los avances computacionales que se han desarrollado es una fuerte y disponible ventaja en la mayoría de las escuelas [20] y como consecuencia, el enfoque de la programación de horarios tiene que tomar en cuenta este fenómeno.

El problema de la programación de horarios puede manifestarse en diversas formas todo dependiendo del entorno en el que se vaya a aplicar, por ejemplo existe la programación de horarios escolares a nivel secundaria, en el cual la mayoría de los alumnos tendrán sus horas corridas y solamente con un periodo libre, llamado receso, que en lo general tiene una duración única de 20 minutos, en este caso, se pretende poder abordar el problema diviéndolo en dos partes, un horario matutino y un horario vespertino.

2.6.1. Problemas P

Un algoritmo es un procedimiento paso a paso para resolver un problema computacional [19]. Para una entrada dada, genera la salida correcta después de un número finito de pasos. La complejidad de tiempo o tiempo de ejecución del algoritmo, expresa el número total de operaciones, como adiciones o comparaciones, como una función del tamaño de la entrada. Se puede decir que un algoritmo es de tiempo polinomial, si su tiempo de ejecución está acotado por un polinomio con variable en el tamaño de entrada.

Entonces, se dice que un algoritmo es eficiente si su tiempo de resolución es polinomial. Los problemas resolubles en tiempo polinomial forman la clase P y están considerados como problemas fáciles de resolver.

La manera matemática de describir este comportamiento es con la notación *«O grande»*, donde para dos funciones $f(n)$, $g(n)$ definidas sobre el mismo dominio de enteros positivos, se escribe $f(n) = O(g(n))$ si existe una constante C tal que $f(n) < Cg(n)$. Esto significa que un algoritmo es polinomial si su tiempo de ejecución, con una entrada de tamaño n, es $O(n^k)$ para algún k.

2.6.2. Problemas NP

La notación NP significa *polinomial no determinístico*. Si se define al algoritmo como no determinístico como un algoritmo que costa de dos fases: suponer y comprobar. Además se sobreentiende que un algoritmo no determinístico siempre hace una suposición correcta.

Para estos problemas, no se conocen algoritmos con tiempo polinomial que los resulevan, y generalmente se cree que tampoco existen, aunque la pregunta ¿P=NP? es decir, si las dos clases de problemas coinciden, es considerada como una de las más importantes en la ciencias computacionales teóricas.

2.6.3. Problemas NP-Completo

Existe otra clase importante de problemas para los cuales no se conocen soluciones eficientes, pero no se ha logrado probar que no la tengan.

Estos problemas se denominan *NP-completos* y tienen la siguiente pecularidad: los mejores algoritmos que se conocen corren en tiempo exponencial, pero se pueden verificar eficientemente. Esto es, si alguien presenta un supuesta solución, se puede diseñar un algoritmo de tiempo polinomial que verifica si en efecto se trata de una solución al problema.

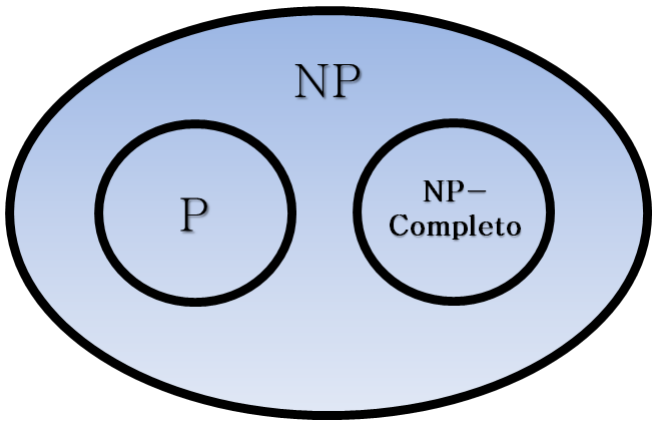

Figura 2.1: Relación entre algunas clases de complejidad

Capítulo 3

Algoritmos Genéticos

Los objetivos de crear inteligencia artificial y vida artificial se remotan a los orígenes de la era de las computadoras. Alan Turing, Jonh von Neumann, Norbert Wiener, etc. estaban, no sólo interesados en la electrónica, sino también en la idea de desarrollar programas de computadora con la inteligencia y con la capacidad adaptativa de aprender y controlar entornos [27]. Así que desde sus primeros pasos, las computadoras estaban también concebidas para modelar el cerebro, imitar el aprendizaje humano y simular la evolución biológica. Estos tres campos han dado lugar a las redes neuronales, el aprendizaje máquina y a la computación evolutiva, cuyo ejemplo más predominante son los algoritmos genéticos.

Los algoritmos genéticos (AG) son «técnicas de búsqueda basadas en la mecánica de la selección natural y la genética» [27, 2, 29]. El material biológico en forma resumida consta de:

> **Célula:** conjunto de cromosomas. En cada célula hay el mismo conjunto de cromosomas.
>
> **Cromosomas:** son cadenas de DNA. Cada cromosoma está constituído por genes.
>
> **Genes:** cada gen codifica a un rasgo particular, por ejemplo el color de ojos. Cada gen tiene su propia posición dentro del cromosoma. Los diferentes rasgos los caracterizan los alelos, por ejemplo ojos azules, marrones, etc.

Los algoritmos genéticos buscan integrar e implementar eficientemente dos ideas fundamentales: las representaciones simples como *cadenas binarias* de las soluciones

del problema y la realización de transformaciones simples para modificar y mejorar estas representaciones.

Para llevar a la práctica el esquema anterior y concretarlo en un algoritmo, hay que especificar los siguientes elementos:

- *Una representación cromosómica.*
- *Una población inicial.*
- *Una medida de evaluación.*
- *Un criterio de selección/eliminación de cromosomas.*
- *Una o varias operaciones de recombinación.*
- *Una o varias operaciones de mutación.*

3.1. Algoritmo genético básico

En la anatomía de un algoritmo genético se pueden observar los siguientes módulos:

1. *Módulo Evolutivo*: mecanismo de codificación (interpreta la información de un cromosoma) y función de evaluación (mide la calidad del cromosoma). Sólo aquí existe información del dominio.
2. *Módulo Poblacional*: tiene una representación poblacional y técnicas para manipularla (técnica de representación, técnica de arranque, criterio de selección y de reemplazo). Aquí también se define el tamaño de la población y la condición de terminación.
3. *Módulo Reproductivo:* contiene los operadores genéticos.

La ejecución de un algoritmo evolutivo requiere de una serie de parámetros de funcionamiento, como por ejemplo el tamaño de la población con la que van a trabajar. Una vez que el algoritmo dispone de los valores para estos parámetros, comienza generando una población de individuos, cada uno de los cuales es un candidato a ser solución del problema en questión o permite llegar a la solución a partir de él. Después se

somete a esa población a un proceso delicado de selección, que modifica la composición de la población, eliminando ciertos individuos y reforzando la presencia de otros, a un proceso de reproducción, que introduce nuevos individuos, y una nueva evaluación, que actualiza los datos de evolución, tales como la adaptación media de la población o la posición del mejor individuo de la población[5].

3.2. Población

Los individuos de la población de un algoritmo genéticos suelen ser cadenas de ceros y unos generadas de forma completamente aleatoria, es decir, se va generando cada gen con una función que devuelve un cero o un uno con igual probabilidad. En algunos problemas en los que se disponga de información adicional que nos permita saber de antemano que determinadas cadenas tienen más probabilidades de llegar a se solución, podemos favorecer su generación al crear la población inicial. Sin embargo, es imprecindible para el buen funcionamiento del algoritmo genético dotar a la población de suficiente variedad para poder explotar todas las zonas del espacio de búsqueda.

3.2.1. Criterio de tamaño de la población

Se debe decidir entre tomar una población muy pequeña (y por tanto convergencia a máximo local) o una población muy grande (y por tanto muchos recursos computacionales). Normalmente se elige una población de tamaño fijo, se puede elegir esquemas de poblaciones de tamaño variable.

3.3. Función de Aptitud (*fitness*)

Es la base para determinar qué soluciones tienen mayor o menor probabilidad de sobrevivir. Se procura tener un balance entre una función que haga diferencias muy grandes (y por tanto una convergencia prematura) y diferencias muy pequeñas (y por tanto un estancamiento).

3.4. Selección

Los individuos son copiados de acuerdo a su evaluación en la función objetivo (*fitness*). Los más aptos tienen mayor probabilidad al contribuir con una o más copias en la siguiente generación (tal cual lo hace la selección natural, en donde sólo los más aptos sobreviven).

Se puede implementar de varias formas, además se puede seleccionar individuos de la población actual, generar una nueva población y reemplazar con ella completamente a la población que se tenía. También a veces se mantienen los N mejores individuos de una población a la siguiente.

3.4.1. Selección elitista

Se garantiza la selección de los miembros más aptos de cada generación (la mayoría de los algoritmos genéticos no utiliza elitismo puro, sino que usan una forma modificada por la que el individuo mejor, o algunos de los mejores, son copiados hacia la siguiente generación en caso de que no surja nada mejor).

3.4.2. Selección proporcional a la aptitud

Los individuos más aptos tienen más probabilidad de ser seleccionados, pero no la certeza.

3.4.3. Selección por ruleta

Una forma de selección proporcional a la aptitud en la que la probabilidad de que un individuo sea seleccionado es proporcional a la diferencia entre su aptitud y la de sus competidores.

3.4.4. Selección escalada

Al incrementarse la aptitud media de la población, la fuerza de la presión selectiva también aumenta y la función de aptitud se hace mas discriminadora. Este método

puede ser útil para seleccionar más tarde, cuando todos los individuos tengan una aptitud relativamente alta y sólo les distingan pequeñas diferencias en la aptitud.

3.4.5. Selección por torneo

Donde se seleccionan 2 individuos aleatoriamente de la población y se opta por el más apto con una probabilidad predeterminada P (y para el menos apto con probabilidad (1-P)).

Se eligen subgrupos de individuos de la población, y los miembros de cada subgrupo compiten entre ellos. Sólo se elige a un individuo de cada subgrupo para la reproducción.

3.4.6. Selección por rango

A cada individuo de la población se le asigna un rango numérico basado en su aptitud, y la selección se basa en este *ranking* en lugar de las diferencias absolutas en aptitud. La ventaja de este método es que puede evitar que individuos muy aptos ganen dominancia al principio a expensas de los menos aptos, lo que reduciría la diversidad genética de la población y podría obstaculizar la búsqueda de una solución aceptable.

3.4.7. Selección generacional

La descendencia de los individuos seleccionados en cada generación se convierte en toda la siguiente generación. No se conservan individuos entre las generaciones.

3.4.8. Selección por estado estacionario

La descendencia de los individuos seleccionados en cada generación vuelven al acervo genético preexistente, reemplazando a algunos de los miembros menos aptos de la siguiente generación. Se conservan algunos individuos entre generaciones.

3.4.9. Selección jerárquica

Los individuos atraviesan múltiples rondas de selección en cada generación. Las evaluaciones de los primeros niveles son más rápidas y menos discriminatorias, mientras

que los que sobreviven hasta niveles más altos son evaluados más rigurosamente. La ventaja de este método es que reduce el tiempo total de cálculo al utilizar una evaluación más rápida y menos selectiva para eliminar a la mayoría de los individuos que se muestran poco o nada prometedores, y sometiendo a una evaluación de aptitud más rigurosa y computacionalmente más costosa sólo a los que sobreviven a esta prueba inicial.

3.5. Operadores Genéticos

En cada nueva generación se crean algunos individuos que no estaban presentes en la población anterior. De esta forma el algoritmo genético va accediendo a nuevas regiones del espacio de búsqueda. Los nuevos individuos se van generando utilizando ciertos «operadores genéticos» a individuos de la población anterior. Los operadores que suelen estar en la mayoría de los algoritmos genéticos son el de cruce y mutación, el primero combina las propiedades de dos individuos para crear nuevos individuos y el segundo crea un individuo nuevo realizando algún tipo de alteración, usualmente pequeña, en un individuo de la población anterior.

3.5.1. Inversión

Tiene baja probabilidad. Incrementa la capacidad de exploración. Permite generar cadenas que serían difíciles de obtener con los otros dos operadores. Opera en un individuo, determina dos posiciones dentro de la cadena e invierte la subcadena.

3.5.2. Elitismo

Se denomina elitismo al proceso por el cual determinados elementos con una adaptación especialmente buena tienen determinados privilegios. Sin embargo, en fases iniciales es peligroso, ya que puede producirse que una élite de superindividuos acabe con la diversidad genética del problema.

3.5.3. Mutación

El operador genético de mutación tiene poca probabilidad de ser utilizado y permite introducir nueva información no presente en la población. Opera sobre un solo individuo, determina una posición y la invierte con cierta probabilidad. Permite salir de máximos locales.

Los principales tipos de mutación son:

3.5.3.1. *Flip Bit*

Es un operador de mutación que invierte simplemente el valor del gen elegido (0 va a 1 y 1 va a 0). Este operador de mutación puede ser utilizado solamente para los genes binarios.

3.5.3.2. Límite (*Boundary*)

Este operador de mutación substituye el valor del gene elegido por el límite superior o más bajo para ese gen (elegido aleatoriamente). Este operador puede ser utilizado solamente para genes codificados como enteros o como valor flotante.

3.5.3.3. No uniforme (*Non-uniforme*)

Este operador de mutación se acerca a la probabilidad de mutación 0 a medida que se incrementa el número de generaciones. Mantiene a la población prácticamente estable en los primeros momentos de la evolución para después permitirle evolucionar al estado final de la solución. Este operador puede ser utilizado solamente para genes codificados como enteros o como valor flotante.

3.5.3.4. Uniforme (*Uniform*)

Este operador de mutación substituye el valor del gen elegido por un valor aleatorio uniforme entre los límites superior e inferior para ese gen. Este operador puede ser utilizado solamente para genes codificados como enteros o como valor flotante.

3.5.3.5. Gausiana (*Gaussian*)

Este operador de mutación agrega una unidad al gen elegido aleatoriamente mediante una distribución gausiana. El nuevo valor del gen se trunca si su valor está fuera del rango permitido para ese gen. Este operador puede ser utilizado solamente para genes codificados como enteros o como valor coma flotante.

3.5.4. Cruce

El operador *cruce* consiste en seleccionar dos individuos después del proceso de selección, determinar una posición de cruce aleatoria e intercambiar las cadenas entre la posición inicial y el punto de cruce y el punto de cruce y la posición final. Tiene una alta probabilidad de ser utilizado y es considerado como el más importante dentro de los Algoritmos Genéticos. Permite la generación de nuevos individuos tomando características de individuos padres.

Existen diferentes tipos de cruce:

3.5.4.1. *Cruce simple*

Utiliza un solo punto de cruce (una máscara de unos seguida de ceros).

3.5.4.2. *Cruce de dos puntos*

Utiliza dos puntos de cruce para la generación de los hijos.

3.5.4.3. *Cruce de n-puntos*

Los dos cromosomas se cortan por n puntos, y el material genético situado entre ellos se intercambia. Lo más habitual es un cruce de un punto o de dos puntos.

3.5.4.4. *Cruce uniforme*

También llamado *razón de mezcla*. En este caso el operador decide con qué tasa (porcentaje) va a particpar la generación de los hijos. Se *genera* un patrón aleatorio de

unos y ceros, y se intercambian los bits de los dos cromosomas que coincidan donde hay un 1 en el patrón. O bien se genera un número aleatorio para cada bit, y si supera una determinada probabilidad se intercambia ese bit entre los dos cromosomas.

3.5.4.5. *Cruces especializados*

En algunos problemas, aplicar aleatoriamente el cruce da lugar a cromosomas que codifican soluciones no válidas; en este caso hay que aplicar el curce de forma que genere siempre soluciones válidas.

3.5.4.6. *Cruce artimético*

Combina linealmente dos pares de cromosomas:

$$H1 = \alpha P1 + (1 - \alpha)P2$$

$$H2 = (1 - \alpha)P1 + \alpha P2$$

donde α es un factor aleatorio de peso.

3.5.4.7. *Cruce heurístico*

Este operador utiliza el valor de adaptación *fitness* de dos pares de cromosomas para determinar la dirección de búsqueda. Los hijos generados formados mediante las siguientes ecuaciones:

$$H1 = Mejor\ Padre + r \cdot (Mejor\ Padre - Peor\ Padre)$$

$$H2 = Mejor\ Padre$$

donde r es un número aleatorio entre 0 y 1.

3.6. Criterio de Parada

Es necesario especificar las condiciones en las que el algoritmo deja de evolucionar y se presenta la mejor solución encontrada. Generalmente viene determinado por criterios *a priori* sencillos, como un número máximo de generaciones o un tiempo máximo de resolución, o más eficientemente por estrategias relacionadas con indicadores del estado de evolución de la población, como por la pérdida de diversidad dentro de la población o por no haber mejora en un cierto número de iteraciones, siendo por lo general una condición mixta lo más utilizado, es decir, limitar el tiempo de ejecución a un número de iteraciones y tener en cuenta algún indicador del estado de la población para considerar la convergencia antes de alcanzar tal limitación.

Capítulo 4

Desarrollo del Modelo

En programación de horarios las estrategias basadas en heurísticas directas involucran el programar todas las sesiones que sean posibles hasta que se encuentren conflictos. En este punto, las sesiones son intercambiadas en un intento de permitir la programación de otra sesión. Generalmente, la mejor estrategia es programar las sesiones de acuerdo a aquella que esté más ajustada.

La elaboración de horarios consiste en asignar de la mejor manera tiempos y lugares a los diferentes cursos que son impartidos en una institución educativa, con el objetivo de satisfacer algunas restricciones importantes que se presenten, como son el número limitado de aulas, de laboratorios, de salas de cómputo, de espacios entre un curso y otro, etc. La principal restricción (central en toda asignación de horarios), es que no existan colisiones, es decir que dos o más cursos que esperan recibir los estudiantes o impartir los profesores no sean asignados simultáneamente.

Para resolver este problema se plantea como uno de coloración de gráficas, en donde cada vértice representa una clase, de las que pueda haber para cada materia, las aristas representan las uniones entre las diferentes clases posibles y las aristas complementarias cuyos extremos representan clases que no pueden ser impartidas en un mismo horario.

Se desean programar todas las asignaturas que imparte la División de Ingenierias Civil y Geomática durante los nueve y ocho semestres respectivamente, que es la duración de cada una de las carreras, esto únicamente en los días y horarios preestablecidos por la coordinación académica, utilizando el problema de coloración robusta generalizado visto en los capítulos anteriores y tomando como referencia a [35], se busca obtener una coloración, en donde a cada vértice, que representa una clase, le será asignado un color.

A la restricción natural que evita la coincidencia en una misma hora de más de una clase de un mismo grupo, correspondientes a la misma asignatura o distintas, se añade la restricción de que dos clases de una misma asignatura no pueden programarse en el mismo día ni en días consecutivos.

4.1. Metodología

Para resolver el problema se parte con base en que la División de Ingenierías Civil y Geomática proporciona la información acerca de las asignaturas que se impartirán en la institución y el número de grupos necesarios, asi como el número de horas que se necesita a la semana para cada una.

Obviamente, el número de clases a impartir en cada una de las horas disponibles, no puede superar al máximo número de salones y tampoco a los 45 intervalos de tiempo con los que se cuenta.

Como se sabe se trata de un problema de minimización, en donde, la función objetivo tendrá a las penalizaciones correspondientes. Lo primero que se va a considerar será la representación de los datos de entrada al algoritmo, no olvidando que se necesita partir por lo más sencillo considerando todos aquellos datos que se relacionan entre sí. Para fines prácticos de este problema los profesores son dejados por el momento aparte de todo el desarrollo, sin embargo, la lista de las asignaturas que obligatoriamente se impartirán semestre con semestre se presentan en las tablas a continuación, correspondiendo la primera a la carrera de Ingeniería Civil y la segunda a Ingeniería Geomática:

Materia	No. veces / semana	No. Grupos	No Clases a impartir	Identificadas como
Semestre 1				
TI	3	3	9	$TI_{11}TI_{21}TI_{31}$ $TI_{12}TI_{22}TI_{32}$ $TI_{13}TI_{23}TI_{33}$
Semestre 2				
TII	3	3	9	$TII_{11}TII_{21}TII_{31}$ $TII_{12}TII_{22}TII_{32}$ $TII_{13}TII_{23}TII_{33}$

Semestre 3				
FMMC	2	2	4	$FMMC_{11}FMMC_{21}$ $FMMC_{12}FMMC_{22}$
MMI	3	3	9	$MMI_{11}MMI_{21}MMI_{31}$ $MMI_{12}MMI_{22}MMI_{32}$ $MMI_{13}MMI_{23}MMI_{33}$
PCE	3	3	9	$PCE_{11}PCE_{21}PCE_{31}$ $PCE_{12}PCE_{22}PCE_{32}$ $PCE_{13}PCE_{23}PCE_{33}$
Semestre 4				
HB	3	3	9	$HB_{11}HB_{21}HB_{31}$ $HB_{12}HB_{22}HB_{32}$ $HB_{13}HB_{23}HB_{33}$
IAMRM	3	3	9	$IAMRM_{11}IAMRM_{21}IAMRM_{31}$ $IAMRM_{12}IAMRM_{22}IAMRM_{32}$ $IAMRM_{13}IAMRM_{23}IAMRM_{33}$
MMII	3	3	9	$MMII_{11}MMII_{21}MMII_{31}$ $MMII_{12}MMII_{22}MMII_{32}$ $MMII_{13}MMII_{23}MMII_{33}$
TGS	2	2	4	$TGS_{11}TGS_{21}$ $TGS_{12}TGS_{22}$
Semestre 5				
AE	3	3	9	$AE_{11}AE_{21}AE_{31}$ $AE_{12}AE_{22}AE_{32}$ $AE_{13}AE_{23}AE_{33}$
GGIA	3	3	9	$GGIA_{11}GGIA_{21}GGIA_{31}$ $GGIA_{12}GGIA_{22}GGIA_{32}$ $GGIA_{13}GGIA_{23}GGIA_{33}$
HMT	3	3	9	$HMT_{11}HMT_{21}HTM_{31}$ $HMT_{12}HTM_{22}HTM_{32}$ $HMT_{13}HMT_{23}HMT_{33}$

IS	3	3	9	$IS_{11}IS_{21}IS_{31}$ $IS_{12}IS_{22}IS_{32}$ $IS_{13}IS_{23}IS_{33}$
Semestre 6				
CS	3	3	9	$CS_{11}CS_{21}CS_{31}$ $CS_{12}CS_{22}CS_{32}$ $CS_{13}CS_{32}CS_{33}$
DE	3	3	9	$DE_{11}DE_{21}DE_{31}$ $DE_{12}DE_{22}DE_{32}$ $DE_{13}DE_{23}DE_{33}$
HC	3	3	9	$HC_{11}HC_{21}HC_{31}$ $HC_{12}HC_{22}HC_{32}$ $HC_{13}HC_{23}HC_{33}$
MT	2	2	4	$MT_{11}MT_{21}$ $MT_{12}MT_{22}$
P	2	2	4	$P_{11}P_{21}$ $P_{12}P_{22}$
Semestre 7				
AGPA	3	3	9	$AGPA_{11}AGPA_{21}AGPA_{31}$ $AGPA_{12}AGPA_{22}AGPA_{32}$ $AGPA_{13}AGPA_{23}AGPA_{33}$
HDIA	3	3	9	$HDIA_{11}HDIA_{21}HDIA_{31}$ $HDIA_{12}HDIA_{22}HDIA_{32}$ $HDIA_{13}HDIA_{23}HDIA_{33}$
MS	3	3	9	$MS_{11}MS_{21}MS_{31}$ $MS_{12}MS_{22}MS_{32}$ $MS_{13}MS_{23}MS_{33}$
ST	3	3	9	$ST_{11}ST_{21}ST_{31}$ $ST_{12}ST_{22}ST_{32}$ $ST_{13}ST_{23}ST_{33}$
APV	3	3	9	$APV_{11}APV_{21}APV_{31}$ $APV_{12}APV_{22}APV_{32}$ $APV_{13}APV_{23}APV_{33}$

PEM	3	3	9	$PEM_{11}PEM_{21}PEM_{31}$ $PEM_{12}PEM_{22}PEM_{32}$ $PEM_{13}PEM_{23}PEM_{33}$
PEECM	3	3	9	$PEECM_{11}PEECM_{21}PEECM_{31}$ $PEECM_{12}PEECM_{22}PEECM_{32}$ $PEECM_{13}PEECM_{23}PEECM_{33}$
Semestre 8				
AI	2	2	4	$AI_{11}AI_{21}$ $AI_{12}AI_{22}$
IP	2	2	4	$IP_{11}IP_{21}$ $IP_{12}IP_{22}$
CMT	3	3	9	$CMT_{11}CMT_{21}CMT_{31}$ $CMT_{12}CMT_{22}CMT_{32}$ $CMT_{13}CMT_{23}CMT_{33}$
HU	3	3	9	$HU_{11}HU_{21}HU_{31}$ $HU_{12}HU_{22}HU_{32}$ $HU_{13}HU_{23}HU_{33}$
OH	3	3	9	$OH_{11}OH_{21}OH_{31}$ $OH_{12}OH_{22}OH_{32}$ $OH_{13}OH_{23}OH_{33}$
TACH	3	2	6	$TACH_{11}TACH_{21}TACH_{31}$ $TACH_{12}TACH_{22}TACH_{32}$
TAR	3	2	6	$TAR_{11}TAR_{21}TAR_{31}$ $TAR_{12}TAR_{22}TAR_{32}$

Tabla 3.1 Identificación de vértices para Ing. Civil

Materia	No. veces/semana	No. Grupos	No. de clases a impartir	Identificadas como:
Semestre 1				
TI	3	3	9	$TI_{11}TI_{21}TI_{31}$ $TI_{12}TI_{22}TI_{32}$ $TI_{13}TI_{23}TI_{33}$

Semestre 2				
TII	3	3	9	$TII_{11}TII_{21}TII_{31}$ $TII_{12}TII_{22}TII_{32}$ $TII_{13}TII_{23}TII_{33}$
Semestre 3				
FI	3	3	9	$FI_{11}FI_{21}FI_{31}$ $FI_{12}FI_{22}FI_{32}$ $FI_{13}FI_{23}FI_{33}$
Semestre 4				
FII	2	2	4	$FII_{11}FII_{12}$ $FII_{12}FII_{22}$
SC	2	2	4	$SC_{11}SC_{21}$ $SC_{12}SC_{22}$
Semestre 5				
FG	2	2	4	$FG_{11}FG_{21}$ $FG_{12}FG_{22}$
MM	2	2	4	$MM_{11}MM_{21}$ $MM_{12}MM_{22}$
TEI	2	2	4	$TEI_{11}TEI_{21}$ $TEI_{12}TEI_{22}$
TIII	3	2	6	$TIII_{11}TIII_{21}TIII_{31}$ $TIII_{12}TIII_{22}TIII_{32}$ $TIII_{13}TIII_{23}TIII_{33}$
Semestre 6				
AP	2	2	4	$AP_{11}AP_{21}$ $AP_{12}AP_{22}$
CGA	2	2	4	$CGA_{11}CGA_{21}$ $CGA_{12}CGA_{22}$
GDIA	3	3	9	$GDIA_{11}GDIA_{21}GDIA_{31}$ $GDIA_{12}GDIA_{22}GDIA_{32}$ $GDIA_{13}GDIA_{23}GDIA_{33}$
GG	2	2	4	$GG_{11}GG_{21}$ $GG_{12}GG_{22}$

HAG	3	3	9	$HAG_{11}HAG_{21}HAG_{31}$ $HAG_{12}HAG_{22}HAG_{32}$ $HAG_{13}HAG_{23}HAG_{33}$
TEII	2	2	4	$TEII_{11}TEII_{21}$ $TEII_{12}TEII_{22}$
Semestre 7				
PRI	2	2	4	$PRI_{11}PRI_{21}$ $PRI_{12}PRI_{22}$
GPS	2	2	4	$GPS_{11}GPS_{21}$ $GPS_{12}GPS_{22}$
SIGI	2	2	4	$SIGI_{11}SIGI_{21}$ $SIGI_{12}SIGI_{22}$
Semestre 8				
PRII	2	2	4	$PRII_{11}PRII_{21}$ $PRII_{12}PRII_{22}$
PG	2	2	4	$PG_{11}PG_{21}$ $PG_{12}PG_{22}$
SIGII	2	2	4	$SIGII_{11}SIGII_{21}$ $SIGII_{12}SIGII_{22}$
APC	3	3	9	$APC_{11}APC_{21}APC_{31}$ $APC_{12}APC_{22}APC_{32}$ $APC_{13}APC_{23}APC_{33}$
CRO	3	3	9	$CRO_{11}CRO_{21}CRO_{31}$ $CRO_{12}CRO_{22}CRO_{32}$ $CRO_{13}CRO_{23}CRO_{33}$
EG	3	3	9	$EG_{11}EG_{21}EG_{31}$ $EG_{12}EG_{22}EG_{32}$ $EG_{13}EG_{23}EG_{33}$
GHM	3	3	9	$GHM_{11}GHM_{21}GHM_{31}$ $GHM_{12}GHM_{22}GHM_{32}$ $GHM_{13}GHM_{23}GHM_{33}$

HDFIA	3	3	9	$HDFIA_{11}HDFIA_{21}HDFIA_{31}$ $HDFIA_{12}HDFIA_{22}HDFIA_{32}$ $HDFIA_{13}HDFIA_{23}HDFIA_{33}$
LT	2	2	4	$LT_{11}LT_{21}$ $LT_{12}LT_{22}$
PGM	3	3	9	$PGM_{11}PGM_{21}PGM_{31}$ $PGM_{12}PGM_{22}PGM_{32}$ $PGM_{13}PGM_{23}PGM_{33}$
TYM	2	2	4	$TYM_{11}TYM_{21}$ $TYM_{12}TYM_{22}$

Tabla 3.2 Identificación de vértices para Ing. Geomática

Tomando la notación de [35] se introducen los 448 vértices

$$TI_{11}, TI_{21}, TI_{31}, TI_{12}, TI_{22}, ..., TYM_{11}, TYM_{21}, TYM_{12}, TYM_{22} \tag{4.1}$$

Que identifican a cada clase X_{ij} por la asignatura $X \in \{TI, TII, FMMC, \ldots, TYM\}$el subídice de la clase asociada $i \in \{1, 2, 3\}$ y el subíndice del $j \in \{1, 2, 3\}$ que se pueden ver con mas detalle en la tabla 4.1 y 4.2 que identifican a cada *clase*.

Y el conjunto de aristas complementarias será aquel que:

- $\{X_{i,j}, Y_{l,m}\}$ verificando que $X = Y$, $i \neq l$ y $j = m$, estas aristas tendrán una fuerte penalización para evitar que tengan el mismo color.
- $\{X_{i,j}, Y_{l,m}\}$ verificando que $X = Y$, $i = l$ y $j \neq m$, estas aristas tendrán una penalización media, ya que se busca una diversidad en los horarios.
- $\{X_{i,j}, Y_{l,m}\}$ verificando que $X \neq Y$, $\forall i, j$, estas aristas tienen una penalización mínima. Con esta penalización se busca que las clases queden distribuidas uniformemente dentro de todas las posibilidades.

El color de cada clase identificará la hora y el día reservados para impartirla. Por lo que habrá un total de 45 colores válidos por el número de salones disponibles.

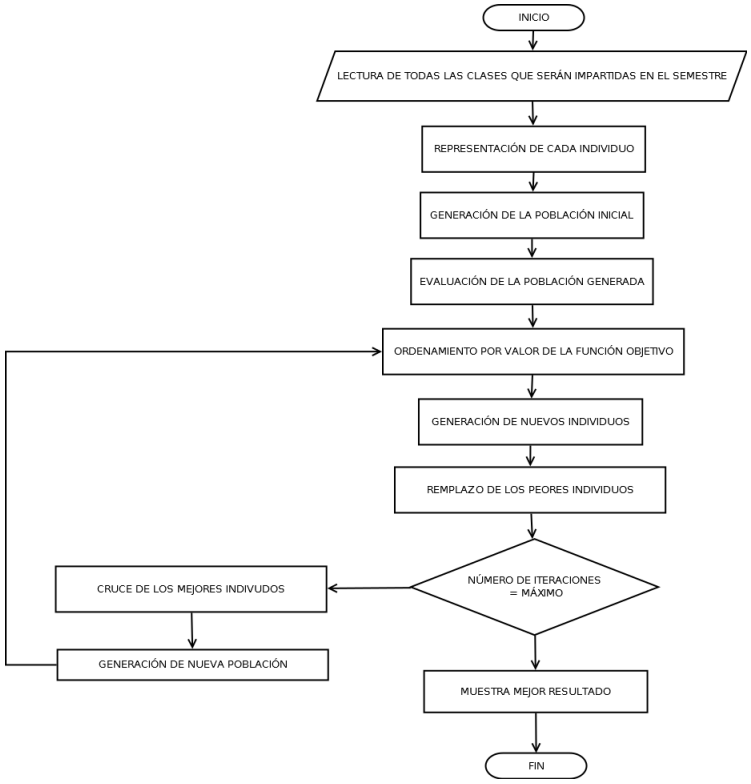

Figura 4.1: Pasos que se llevarán a cabo para la metodología

4.2. Representación de los individuos

La simplicidad que da el uso de los algoritmos genéticos es una característica importante, y es necesario disponer de un método para la correcta interpretación de cada uno de los individuos que se tendrán, además de que es necesario para poder evaluar la adaptación del individuo como solución al problema.

Debemos buscar una codificación tal que cada punto del espacio de búsqueda esté representado por el mismo número de cadenas binarias, y tal que sea capaz de representar todos los puntos del espacio del problema. [5]

Los individuos serán representados por la asignación de horario que tendrán de acuerdo a la asignatura, grupo y clase a la que pertenecen, por lo que, los individuos serán listas

de genes en donde cada gen representa el color que le fue asignado para la impartición de dicha clase.

El color de cada clase identificará la hora y el día reservados para impartirla. Habrá por lo tanto 45 colores válidos por el número de salones que se designen.

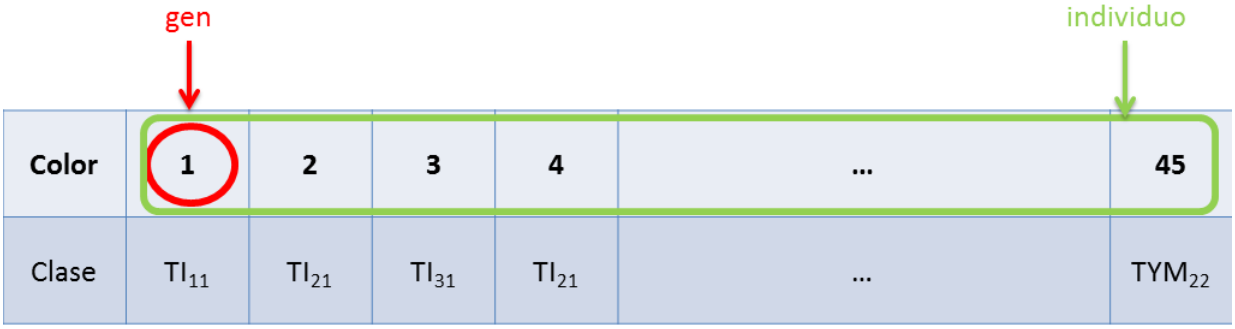

Figura 4.2: Ejemplo de Individuo

Según la figura 4.2 cada posición de la cadena del individuo tiene un significado para el problema, el cual es el color que le corresponde, y con ello se sabe cual es el horario por defecto en donde tendrá lugar la impartición de la clase, este individuo además de cumplir con las restricciones propuestas tiene que tener un valor bajo en la evaluación de la función objetivo, con el fin de garantizar que al cruzarlo con otro individuo bien evaluado obtendremos una mejora en la población en general.

4.3. Generación de la población inicial

La solución se representa asignándoles a cada uno de los vértices un color que representa la hora y el día en el que será impartida esa clase, los colores como ya lo vimos en los puntos anteriores están ubicados en horarios designados de acuerdo a la política que marca la Secretaría Académica de la División.

Lo primero será tener una coloración de los vértices en algún color, una posible solución sería poder colocarlas en órden, esto es, el vértice 1 en el color 1, el vértice 2 en el color 2, etc., sin embargo, esto provocaría que hubiera conflictos en su correcta impartición, además de que el arreglo que se tendría no sería satisfactorio.

Por lo que para la población inicial, se asegurará de tener una coloración válida de todos los vértices aunque evaluada en la función objetivo tenga un valor muy grande, se sabe

que se disminuye la penalización con el algoritmo.

Por lo que se genera una solución posible y se mejorará al paso del algoritmo propuesto, esta solución se crea en función de que cumpla con el grupo de restricciones comunes: La clase A_{11}no puede impartirse a la misma hora que la clase B_{11} y también la clase A_{11} tiene que tener d días entre la clase A_{12}.

Teniendo en cuenta con cuántos véritces y aristas esta conformada la gráfica, se debe asegurar que se respetarán las coloraciones, por tanto, se penalizará fuertemente aquellas aristas complementarias que unan a los vértices de la misma asignatura y diferentes clases. *Una restricción importante es que aunque algunos de los vértices pueden llegar a tener el mismo color, este número de color no puede exceder al número de salones disponibles para impartir la clase en ese mismo horario* [5].

Observando la siguiente matriz 4.3 cuadrada, de dimensión $n\,x\,n$ que almacena en la matriz triangular inferior la matriz de penalizaciones y en la matriz superior la matriz de adyacencias.

-	GTIA_11	GTIA_21	GTIA_31	GTIA_12	GTIA_22	GTIA_32	GTIA_13	GTIA_23	GTIA_33	EE_11	EE_21	EE_31	EE_12	EE_22	EE_32	EE_13	EE_23	EE_33	EP_11	EE_21	EP_12	EP_22	PO_11	PO_21	PO_31	PO_12
GTIA_11	-	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GTIA_21	100	-	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GTIA_31	100	100	-	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GTIA_12	50	-	-	-	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GTIA_22	-	50	-	100	-	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GTIA_32	-	-	50	100	100	-	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GTIA_13	50	-	-	50	-	-	-	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GTIA_23	-	50	-	-	50	-	100	-	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GTIA_33	-	-	50	-	-	50	100	100	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EE_11	1	1	1	1	1	1	1	1	1	-	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0
EE_21	1	1	1	1	1	1	1	1	1	100	-	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0
EE_31	1	1	1	1	1	1	1	1	1	100	100	-	1	1	1	1	1	1	0	1	0	0	0	0	0	0
EE_12	1	1	1	1	1	1	1	1	1	50	-	-	-	1	1	1	1	1	0	1	0	0	0	0	0	0
EE_22	1	1	1	1	1	1	1	1	1	-	50	-	100	-	1	1	1	1	0	1	0	0	0	0	0	0
EE_32	1	1	1	1	1	1	1	1	1	-	-	50	100	100	-	1	1	1	0	1	0	0	0	0	0	0
EE_13	1	1	1	1	1	1	1	1	1	50	-	-	50	-	-	-	1	1	0	1	0	0	0	0	0	0
EE_23	1	1	1	1	1	1	1	1	1	-	50	-	-	50	-	100	-	1	0	1	0	0	0	0	0	0
EE_33	1	1	1	1	1	1	1	1	1	-	-	50	-	-	50	100	100	-	0	1	0	0	0	0	0	0
EP_11	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-	0	1	1	0	0	0	0
EE_21	1	1	1	1	1	1	1	1	1	100	-	100	-	50	-	-	50	-	1	-	0	0	0	0	0	0
EP_12	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	50	1	-	1	0	0	0	0
EP_22	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-	1	100	-	0	0	0	0
PO_11	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-	1	1	1	1
PO_21	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	100	-	1	1	1
PO_31	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	100	100	-	1	1
PO_12	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	50	-	-	-	-
PO_22	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-	50	-	100	100
PO_32	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-	-	50	100	100
PO_13	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	50	-	-	50	50
PO_23	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-	50	-	-	-
PO_33	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-	-	50	-	-

Figura 4.3: Matriz de datos

La coloración obtenida con un algoritmo genético es la que mejorará la coloración.

1. **Se genera la población inicial de 200 individuos para obtener una primera solución.**

   Esta población se genera de manera pseudoaleatoria, lo que quiere decir que se van generando individuos que siempre cumplan con las restricciones impuestas,

por lo tanto, las soluciones satisfarán cada una de las condiciones. El tamaño de la población será de 200 individuos.

El tiempo que se requiere para obtener el resultado de la función objetivo, depende también, del número de población que se maneje así como de el número de iteraciones que se tome como criterio de parada, para fines de este apartado se hablará solamente de como se va a tratar el resultado de esta función objetivo.

En este caso se toma al 80 % de la población como los mejores genes y se desecha el resto, ya que no entra en la categoría de mejores padres para generar a la siguiente generación. Es entonces cuando se toma este 80 % de población y se va a cruzar para obtener nuevos elementos que reemplacen el 20 % que fué desechado y obtener mejores individuos de la población. El mismo procedimiento se repetirá hasta que el criterio de parada se haya cumplido.

4.4. Modelo de programación matemática

Para el modelo de programación matemática propuesto se considera una gráfica G que tiene un conjunto de vértices V, un conjunto de aristas A y un conjunto de aristas complementarias $\bar{A}$. Además de definir con claridad la función objetivo que en su sentido más general está denotado por:

$$Min \quad R(C) = \sum P_{ij} \tag{4.2}$$

Es decir, obtener el mínimo de la suma de las penalizaciones para lograr que dos vértices que tengan aristas complementarias altamente penalizadas no tengan el mismo color y además se encuentren a una distancia máxima determinada. Por lo que, el PCRG *[36, 41]* introduce la definición de distancia entre los colores, la cual esta definida para cada par de vértices que se encuentran unidos por una arista complementaria, además se sabe que el PCRG es flexible al permitir definir diferentes distancias entre los colores, lo cual es de mucha utilidad dada la complejidad de este problema en particular, ya que, se repite sistemáticamente e interesa que de acuerdo a las asignaciones iniciales que son las primeras horas del día, los colores 1,2,..., sean incompatibles con las asignaciones de las últimas horas, los colores, c-2, c-1, c, ..., por lo que una cota $\hat{d}$ se determinaría

mediante una c-distancia circular de una matriz M. Entonces con el fin de incorporar la *c-distancia* entre colores

$$d : \{1, 2, ..., c\} \times \{1, 2, ..., c\} \to \{0, 1, 2, ...\} \tag{4.3}$$

Y el límite o umbral $\hat{d}$ en el modelo de programación matemática del $PCRG$, se introduce la matriz M de dimensión $c \times c$ definida por:

$$m_{kk'} = \begin{cases} 1 & d(k, k') > \hat{d} \\ 0 & d(k, k') \leq \hat{d} \end{cases} \quad \forall k, k' \in \{1, ..., c\} \tag{4.4}$$

Por ejemplo, fijando c=4, si la 4-distancia es d^0, la distancia trivial y el umbral es $\hat{d} = 0$, la matriz M es:

$$M = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \tag{4.5}$$

Con la 4-distancia d^1basada en el valor absoluto de la diferencia, con $\hat{d} = 0$ y $d \hat{=} 1$ se tienen las siguientes matrices:

$$M = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \qquad M' = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix} \tag{4.6}$$

Una vez incorporados los datos de la c-distancia y la cota $\hat{d}$ en la matriz M, el PCRG queda caracterizado por la gráfica $G = (V, A)$, el número de colores válido c y las penalizaciones $\left\{p_{ij}, \{i, j\} \in \bar{A}\right\}$.

Las variables de decisión asociadas a la coloración C^c están definidas por:

$$x_{ij} = \begin{cases} 1 & C^c(i) = k \\ 0 & C^c(i) \neq k \end{cases} \qquad i \in \{1, ..., n\}, k \in \{1, ..., c\} \tag{4.7}$$

Las restricciones del modelo de programación son las siguientes:

- La coloración debe estar bien definida en todos los vértices

$$\sum_{k=1}^{c} x_{ik} = 1 \qquad \forall i \in \{1, ..., n\} \tag{4.8}$$

- La coloración válida

$$x_{ik} + x_{jk} \leq 1 \qquad \forall \{i, j\} \in A \qquad \forall k \in \{1, .., c\} \tag{4.9}$$

Con el fin de indentificar las aristas complementarias que unen vértices con coloraciones suficientemente cercanas, se introducen las variables auxiliares:

$$y_{ij} = \begin{cases} 1 & si\ d(C^c(i), C^c(j)) \leq \hat{d} \quad \{i, j\} \in \bar{A} \\ 0 & si\ d(C^c(i), C^c(j)) > \hat{d} \quad \{i, j\} \in \bar{A} \end{cases} \qquad \forall \{i, j\} \in \bar{A} \tag{4.10}$$

Es decir , $y_{ij} = 1$ si y sólo si los extremos de la arista complementaria $\{i, j\} \in \bar{A}$, con colores $k = C^c(i)$ y $k' = C^c(j)$, verifican $m_{kk'} = 0$.

Con el fin de garantizar esta propiedad se introduce, por un lado, la siguiente familia de restricciones lineales:

$$x_{ik} + \sum_{k'=1}^{c} x_{jk'}(1 - m_{kk'}) \leq y_{ij} + 1 \quad \forall \{i, j\} \in \bar{A}, \quad \forall k \in \{1, ..., c\} \tag{4.11}$$

de forma que existen dos colores k, k' suficientemente próximos, $m_{kk'} = 0$, que colorean los extremos de una arista complementaria $\{i, j\} \in \bar{E}$, entonces el valor de las variables auxiliares son $y_{ij} = 1$.

Por otro lado, al considerar la función objetivo que penaliza las aristas complementarias con colores suficientemente próximos

$$R^{\hat{d}}(C^c) = \sum_{\{i,j\} \in \bar{A}} p_{ij} y_{ij} \tag{4.12}$$

al ser $p_{ij} \geq 0$ garantiza el valor $y_{ij} = 0$ excepto cuando i y j sean los extremos de aristas complementarias con colores suficientemente próximos.

Si en este modelo se hace $\hat{d} = 0$, entonces se tiene el modelo de programación binaria para el PCR, lo que demuestra que el PCR es un caso particular del PCRG. En efecto, si $\hat{d} = 0$ no es necesario definir la matriz M y las variables y_{ij} valdrán uno si se usa el color k y cero en otro caso.

El problema de coloración robusta generalizado es muy flexible al permitir definir diferentes distancias entre los colores. En el problema de un horario diario, por ejemplo, que se repite sistemáticamente, puede interesar que las asignaciones de las primeras horas del día, los colores 1,2,..., sean incompatibles con las asignaciones de las últimas horas, los colores . . ., c - 2, c - 1, c, por lo que una *c-distancia* circular determinaría para una cota fija $\hat{d}$ una matriz M del tipo

$$M = \begin{bmatrix} 0 & 0 & 0 & \cdots & 1 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 1 & 1 & 0 \\ 0 & 0 & 0 & \cdots & 1 & 1 & 1 \\ \\ \vdots & & & \vdots & & & \vdots \\ \\ 1 & 1 & 1 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 1 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 & 0 \end{bmatrix} \tag{4.13}$$

La matriz M permite identificar situaciones más generales que las contempladas por una *c-distancia* definida en el conjunto de colores y una cota o umbral $\hat{d}$.

Para este trabajo, se fijará $\hat{d}$*=18*, y se utiliza la distancia d^1 para evitar que una asignatura se imparta en dos dias consecutivos, el primero a última hora y el siguiente a primera hora. En consecuencia el modelo impuesto garantizará las restricciones del problema aunque es más restrictivo.

Definiendo de manera apropiada la matriz M, se puede obtener un modelo menos restrictivo que el anterior y verificando todas las restricciones originales del problema.

Esta matriz M que tendría la siguiente estructura:

$$M = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & \cdots & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & & 1 & 1 & 1 & 1 & 1 & 1 \\
\vdots & & & & & & \vdots & & & & & & \vdots \\
1 & 1 & 1 & 1 & 1 & 1 & & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & \cdots & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix} \tag{4.14}$$

La gráfica es de 448 vértices, los asociados a las clases de las asignaturas, tiene 8 componentes conexas, una por semestre. Cada componente conexa tiene tantos vértices como clases y todos los vértices de las clases de cada asignatura del curso están unidos al resto de los vértices de las otras asignaturas; no están unidos los vértices correspondientes a clase de una misma asignatura.

Las aristas complementarias se penalizan de forma diferente según su origen:

- Las aristas complementarias uniendo clases de una misma asignatura se penalizan con una cantidad suficientemente grande; por ejemplo, $p_{X_{ij}X_{kj}} = 100$ para todo par de clases de una misma asignatura.
- Las aristas complementarias que unen clases de mismas asignaturas de distintos grupos se penalizan con una cantidad inferior a la anterior; por ejemplo, $p_{X_{ij}X_{kj}} = 50$ para cualquier par de clases.
- Las aristas complementarias que unen clases de asignaturas de distintos semestres se penalizan con una cantidad positivamente incomparablemente inferior a la primera; por ejemplo, $p_{X_{ij}X_{kj}} = 1$ para cualquier par de clases.

Por lo que el modelo de programación quedaría

$$Min\ R^d(C^c) = \sum 100y_{i,j} + \sum 50y_{k,l} + \sum y_{m,n} \tag{4.15}$$

$$s.a. \quad \sum_{k=1}^{c} x_{ik} = 1 \qquad \forall i \in \{1, ..., n\} \tag{4.16}$$

$$x_{ik} + x_{jk} \leq 1 \qquad \forall \{i, j\} \in A \qquad \forall k \in \{1, .., c\} \tag{4.17}$$

$$x_{ik} + \sum_{k'=1}^{c} x_{jk'}(1 - m_{kk'}) \leq y_{ij} + 1 \quad \forall \{i, j\} \in \bar{A}, \quad \forall k \in \{1, ..., c\} \tag{4.18}$$

4.4.1. Mejoramiento de la población inicial.

El siguiente pseudocódigo muestra el algoritmo que se utiliza para ir mejorando la población inicial. Este algoritmo cumple con las características de un algoritmo genético, los cuales han dado buenos resultados en este tipo de aplicaciones [6, 38, 3, 9].

Algoritmo 4.1 Pseudocódigo del algoritmo genético

```
Sea la gráfica G = (V,A) donde V es el conjunto de vértices y A es el conjunto de
    aristas
REPETIR
        Generar elemento de la población de manera aleatoria
        Calificar elemento de la población por medio de la F.O.
        Agregar elemento a la población
HASTA Total de la población
REPETIR
        Ordenar población por resultado de la F.O.
        REPETIR
                Cruzamos los mejores elementos de la población generando un nuevo
                    elemento
                Calificar nuevo elemento por medio de F.O.
                Sustituir el peor elemento de la población con el nuevo elemento
        HASTA Sustituir todos los peores elementos de la población a eliminar
HASTA Total de iteraciones
Mostrar resultado
```

4.4.2. Evaluación de la población generada

Se evalúa a cada individuo de la población con la función de aptitud *"fitness"*, que en este caso será mediante la función objetivo. 4.15. Esto porque después de cada generación se revisan los contadores de adaptación relativa y puntuación acumulada

de los individuos de la población. Así mismo, se calcula la adaptación global de la población y la posición del mejor individuo. El siguiente es un fragmento de codigo que se utiliza para la evaluación ya implementada en el algoritmo.

Algoritmo 4.2 Pseudocódigo evaluación de la población generada

```
FUNCION Evaluar_Poblacion {
        Para cada individuo de la poblacion
                Evalua al individuo
                Si esta definido revisa la penalizacion
                Ind->{Penalizacion}=0;
                REPETIR desde i = 1 hasta N
                        REPETIR desde j = 0 hasta i
                                SI Ya existe un color igual para el gen
                                        Suma la penalizacion;
                                Fin si
                        HASTA El numero total de genes
                HASTA El numero total de individuos
        HASTA El numero total de poblacion
        obj{Evaluado}=1;
        REGRESA obj{Poblacion};
FIN FUNCION
```

4.4.2.1. Ordenamiento de las soluciones para la formación de nuevas poblaciones

Una vez que la función de aptitud evaluó a los individuos de la población, se ordenan de mejor a peor y se separa al 20 % superior y al 20 % inferior, a estos últimos se les va a retirar de la población, y los primeros se van a tomar para generar nuevos individuos. La decisión de tomar el 20 % se basa en el principio de Pareto, también conocido como la regla del 80-20, que según Vilfredo Pareto la gente en su sociedad se dividía naturalmente entre los «pocos de mucho» y los «muchos de poco»; se establecían así dos grupos de proporciones 80-20 tales que el grupo minoritario, formado por un 20 % de población, ostentaba el 80 % de algo y el grupo mayoritario, formado por un 80 % de población, el 20 % de ese mismo algo. Estas cifras son arbitrarias; no son exactas y pueden variar. Su aplicación reside en la descripción de un fenómeno y, como tal, es aproximada y adaptable a cada caso particular.

4.4.2.2. Generación de nuevos individuos

Para mantener el número de población establecido desde el principio, se generan nuevos individuos haciendo cruces entre el 20 % superior de individuos.

4.4.2.3. Reemplazo de los peores individuos

Con la nueva generación creada, se añaden a la población, sustituyendo al 20 % inferior.

4.4.2.4. Definición del criterio de parada.

Los pasos anteriores se repetirán hasta llegar al momento que se ha definido como criterio de parada, el cual para este caso se refiere al número de iteraciones.

4.4.2.5. Representación de una buena solución.

Después de que el algoritmo ha terminado su ejecución, mediante el resultado de la función objetivo, se determina cual es la mejor solución y se toma como resultado final.

4.5. Operador de cruce uniforme

A cada solución se le va a mejorar con el paso del algoritmo, esto se refiere a que se tomará una primera coloración como parte de una población inicial, la población inicial será de 200 individuos, y se generará una nueva solución eliminando el 20 % de los individuos menos aptos y se generarán nuevos individuos con el cruce del 20 % de los mejores individuos que serán los padres para los nuevos elementos y formar así una nueva población que cumpla con todas las restricciones y que mejore el valor de la función de aptitud, que será la función objetivo. Se utiliza un cruce especializado que está basado en el cruce uniforme, por ejemplo:

- *Se genera una cadena aleatoria de 0 y 1 de tamaño n, la probabilidad de que sea 1 es p.*

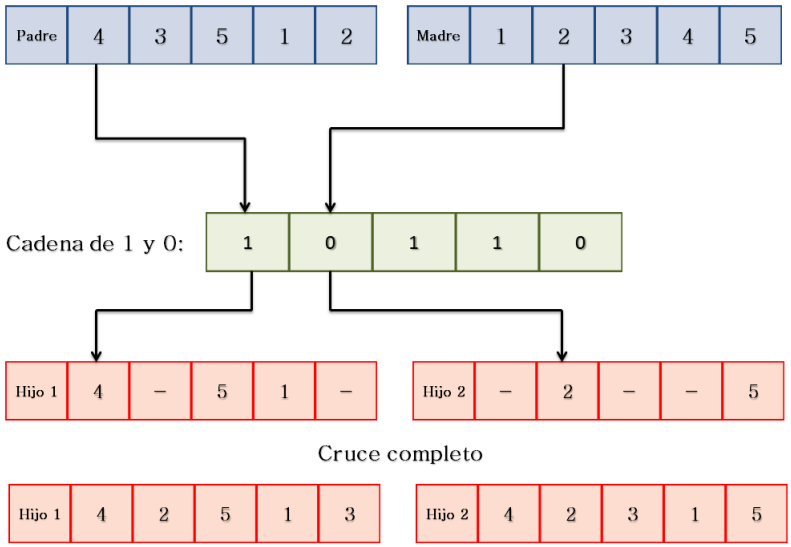

Figura 4.4: AG - Cruce uniforme

- *Se generan dos hijos de la siguiente manera*:
  - *El hijo 1*
    - *Se genera colocando en la posición en la que haya un 1 en la cadena el elemento que ocupe esa posición en el padre.*
    - *Hacer una lista con los elementos del padre asociados con un 0 en la cadena.*
    - *Permutar los elementos para que aparezcan en el mismo orden en que aparecen en la madre.*
    - *Llenar los huecos con los números en el orden generado en* $\boldsymbol{c}$.
  - *El hijo 2*
    - *Colocando en la posición en que haya un 0 en la cadena, el elemento que ocupe esa posición en la Madre.*
    - *Hacer una lista con los elementos de la madre asociados con un 1 en la cadena.*
    - *Permutar los elementos para que aparezcan en el mismo orden en que aparecen en el padre.*
    - *Llenar los huecos con los números en el orden generado en* $\boldsymbol{g}$.

- Se verifica que cada solución tenga una coloración válida, de no ser así, se generará una mutación en el gen que no cumpla con los criterios de validación. El criterio de paro será de acuerdo al número de iteraciones, y se tomará el último y mejor resultado obtenido y se mostrará como solución final.

El cruzamiento como ya se había hablado simula la descendencia de ambos padres y heredan sus genes a sus hijos, en realidad lo que se está llevando a cabo es un mecanismo de recombinación. La manera en que se lleva a cabo en este caso es, se seleccionarán dos padres, de los mejores resultados, aleatoriamiente.

Cuando ambos padres fueron electos, se empezarán a cruzar sus genes para dar *vida* al hijo, esto es mediante un cruzamiento uniforme, que significa que mediante la utilización de una cadena de 1's y 0's, se sabría si el *gen* que se utilizará en la formación de un nuevo individuo será de la mamá o del papá, si es 1 será del padre y si es 0 será de la madre (Figura 4.4). Se considera especializado ya que no se permite tener hijos que no cumplan con alguna de las restricciones que ya están planteadas ya que el naturaleza del problema impide que algunas de las asignaciones de color que pueden resultar de la mezcla de *genes* de los padres, sea totalmente inválida de aplicar en particular a nuestro problema real.

Si esto ocurre, es decir, si el *gen* elegido no cumple con las restricciones, entonces se ocupara el *gen* del otro padre, si tampoco cumpliera con las restricciones entonces damos otro color de forma aleatoria hasta que obtengamos un color que si pueda resultar válido, cuando terminamos la formación del hijo, entonces se vuelve parte de una nueva población y todo el proceso se repite para obtener todos los individuos necesarios.

Algoritmo 4.3 Pseudocódigo para el cruce de individuos

```
Funcion Cruzar_Individuos {
        Termina si el numero de individuo padre a cruzar no definido

        Termina si el numero de individuo madre a cruzar no definido

        Muere si los individuos no estan generados
                sino definelo en Poblacion
        Muere si la matriz de restricciones y penalizaciones no estan defin
                sino definelo en M_Restricciones_Penalizaciones
        Obten el numero maximo de clases por color
        Obtener al padre
        Obtener a la madre

        Realizar la mascara para el cruce
        PARA i = 0 hasta i < N incrementos 1
                selecciona para cada elemento de la mascara
                un numero aleatorio entre 0 y 1
        Empezamos el cruce por medio de la mascara
        Revisando que cada color nuevo sea valido.
        En caso contrario se utilizara el del otro padre
        Y si sigue siendo valido se generara
        Aleatoriamente hasta que sea valido.
        Si sigue siendo invalido despues de muchos intentos
        Se genera una nueva mascara de cruce y se empieza de nuevo
        Si se excede el numero maximo de intentos esperado se crea otro ind
        Si el color es valido aumentamos el contador de ese color
Termina la funcion
```

4.6. Restricciones adicionales

Se puede considerar otro tipo de restricciones para este mismo problema, por ejemplo, considerar más elementos que también son importantes en la planificación de los horarios.

4.6.1. Restricciones generales

1. Un profesor solo puede dar una clase en un horario a la vez.

2. Un salón solo puede ser utilizado por una clase en un horario a la vez.

4.6.2. Restricciones específicas

1. Dos clases de la misma asignatura y del mismo grupo no deben ser programadas en el mismo día.

2. Las clases solo pueden iniciar a partir de las 7:00 am y deben terminar antes de las 10:00 pm.

3. Se dejó una franja de disponibilidad de 15:00 a 16:00 hrs.

Capítulo 5

Pruebas y validación del modelo

Reuniendo todos los datos de entrada necesarios, se empezó a ejecutar el algoritmo incorporando una por una las restricciones que ya se tenían definidas, esto debido a que se necesitaba saber cual sería el comportamiento del mismo, es claro, que muchas veces no se sabe cual sea el resultado práctico de lo que en teoría se tiene bien estudiado, sin embargo conforme si iban incorporando las restricciones, los resultados tomaban una forma real y tangible, es decir, los resultados mostraban que se podía contar con una solución totalmente implementable.

Gracias a otros trabajos bien fundamentados como [6, 35, 18] se logra desarrollar un planteamiento más claro, en particular en [1] se toma la misma base del PCRG con búsqueda Tabú pero con la propuesta de utilizar algoritmos genéticos, lo que se ha implementado en este trabajo.

5.1. Resultados Computacionales

El criterio de parada como ya se ha mencionado es el número de iteraciones, sin embargo para elegir este número de iteraciones, se procede a programar el algoritmo en algún lenguaje de programación y con su respectiva ejecución en la computadora, se tienen elementos para poder decidir cuál sería el número adecuado de iteraciones y también el de la población, ya que, no se puede elegir arbitrariamente entre tomar una población muy pequeña (y por tanto convergencia a máximo local) y una población muy grande (y por tanto muchos recursos computacionales) para este problema. Así pues, también se llevaron algunos estudios para el número de iteraciones.

Después de varias corridas, con diferentes números de iteraciones, como por ejemplo 100, 150, 200, 300, 500, entre otras, y un tamaño de población de entre 100 a 400 se muestran en seguida las más representativas y que fueron decisivas para la obtención de un buen valor de la función objetivo.

5.1.1. Elección del número de iteraciones

Como se deben investigar mas de dos parámetros que son de importancia para la convergencia y obtención de una buena solución, se pondrá, en primera instancia, un número fijo para el tamaño de la población, esto es en 100. Entonces se procederá a realizar varias corridas del algoritmo para encontrar cual sería un número representativo para las iteraciones.

Como se pude ver en la figura 5.1, se ha ejecutado el algoritmo varias veces con un criterio de parada de 200 iteraciones. Con el fin de determinar si este criterio es bueno para encontrar buenos resultados.

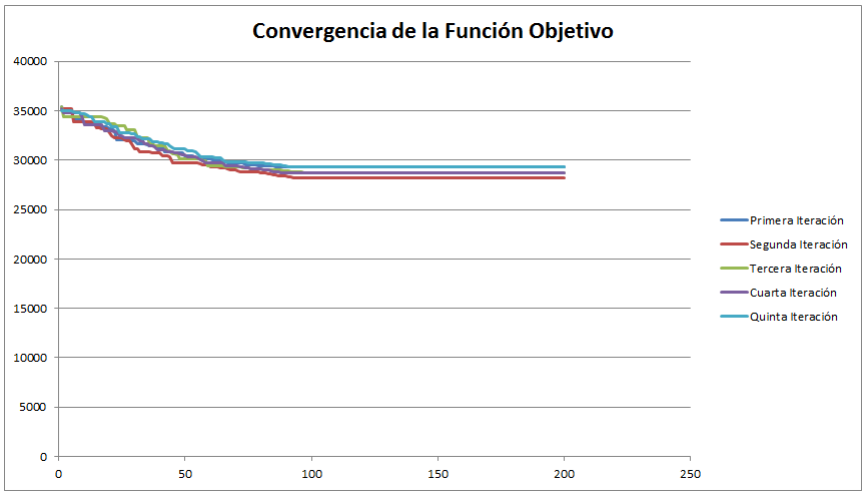

Figura 5.1: Criterio de parada de 200 Iteraciones, Población de 100

Se aprecia que el número de iteraciones donde converge la gráfica es cerca de las 100 iteraciones. En el algoritmo tambien se incluyó un benchmark para evaluar el tiempo que tarda el algoritmo mostrado en la siguiente gráfica.

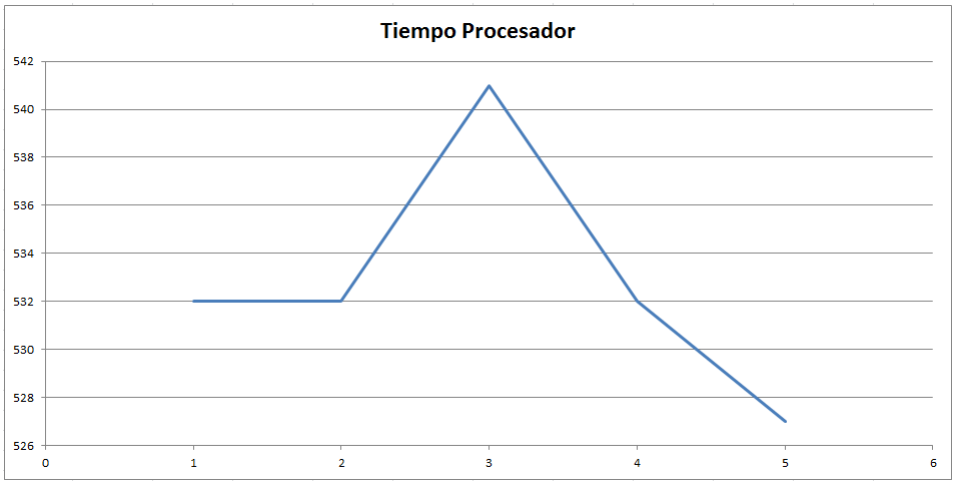

Figura 5.2: Benchmark para 200 Iteraciones Población 100

En la figura 5.3, con un criterio de parada a las 150 iteraciones de cada una de las corridas dejando la población en 100.

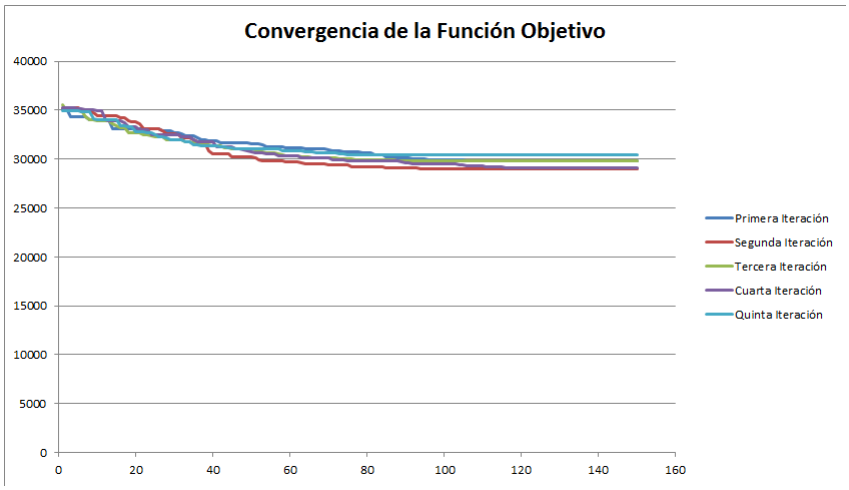

Figura 5.3: Criterio de parada de 150 Iteraciones, Población de 100

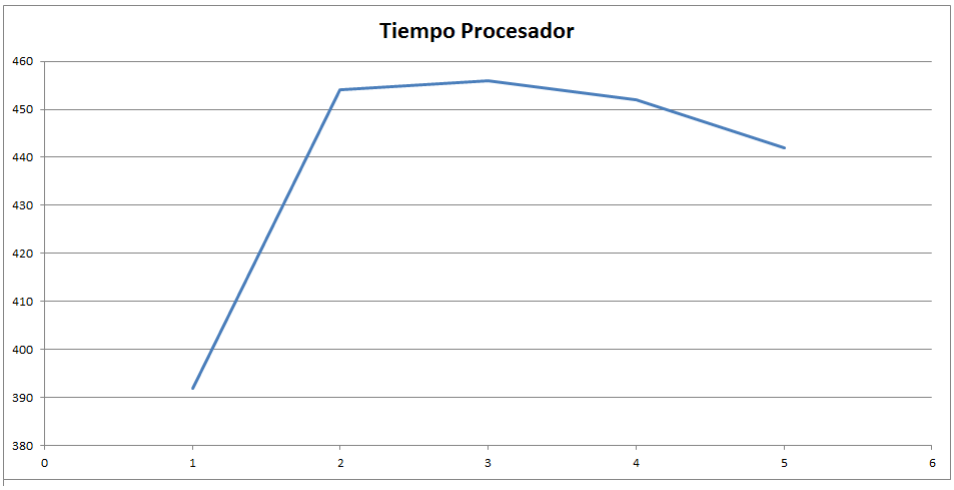

Figura 5.4: Benchmark para 150 Iteraciones Población 100

En la figura 5.3, con un criterio de parada a las 100 iteraciones de cada una de las corridas dejando la población en 100.

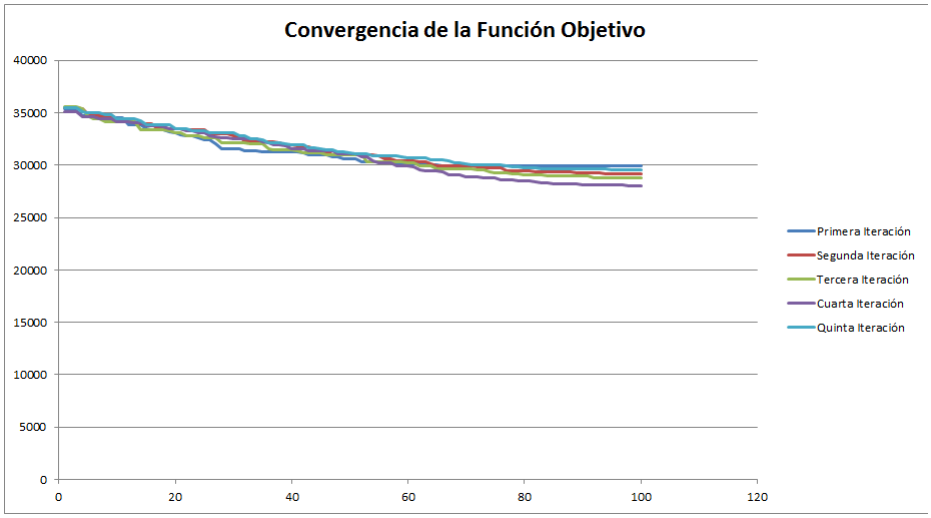

Figura 5.5: Criterio de parada de 100 Iteraciones, Población de 100

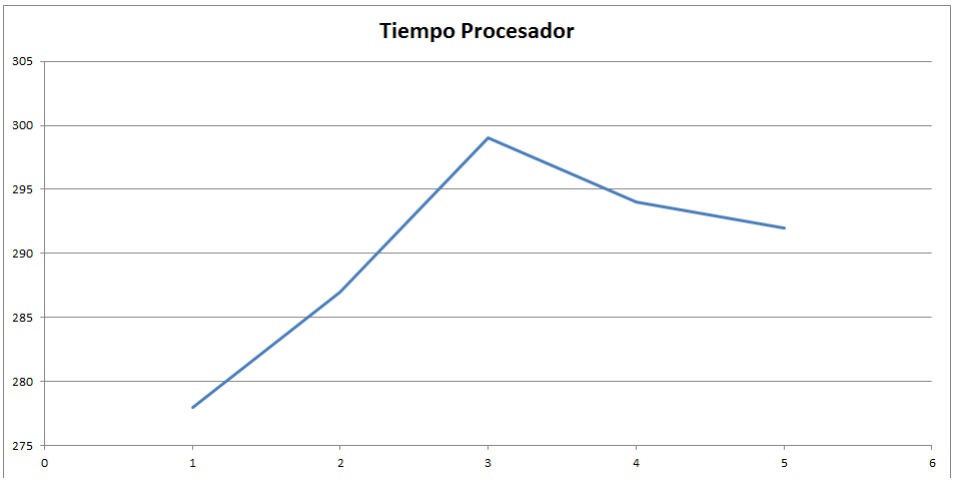

Figura 5.6: Benchmark para 100 Iteraciones Población 100

Se aprecia que el número de iteraciones donde converge nuestra gráfica y la función objetivo es nuevamente cerca de las 100 iteraciones, por lo que podriamos asumir que un buen número de iteraciones sería 100, sin embargo todavía faltaría mover los parámetros de población, cruce y eliminación para poder decir que tenemos una buena solución.

Con base a lo anterior, se concluye que el aumentar mucho el número de iteraciones en realidad no implica un cambio significativo, ya que la convergencia de la función objetivo siempre cae siempre muy cerca de las 100 iteraciones lo que conlleva a deducir que lo que se tiene que variar sería el tamaño de la población para aumentar el número de individuos.

5.1.2. Elección del tamaño de la población

Siguiendo con la misma vertiente de fijar uno de los dos parámetros buscados y teniendo que, ya se cuenta con el número indicado para el criterio de parada, en este caso de 100, se procederá a realizar varias corridas del algoritmo para encontrar cual sería un número representativo para el tamaño de la población.

Como se puede ver en la gráfica 5.7, se ha ejecutado el algoritmo, con un criterio de parada de 100 y un tamaño de población de 50 para cada una de las corridas.

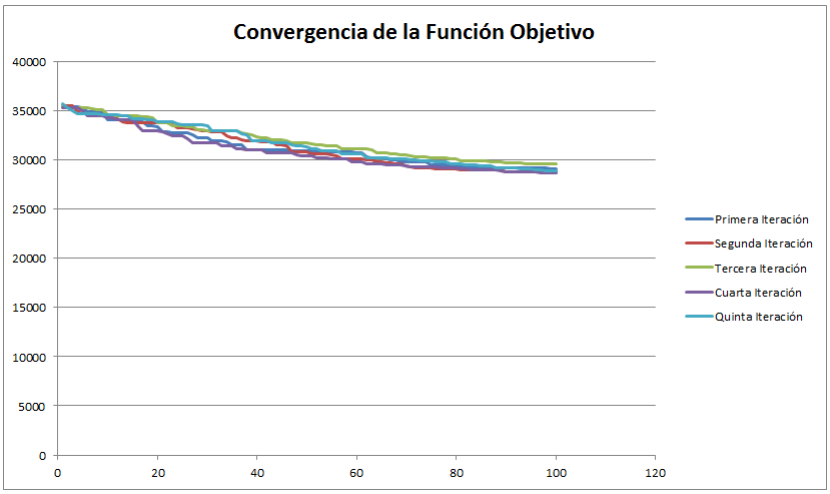

Figura 5.7: Criterio de parada de 100 Iteraciones, Población de 50

Y podemos concluir que en realidad no se estabiliza a las 100 iteraciones como era de esperarse, sino que no podruce una buena solución, asi que aumentaremos el número de población.

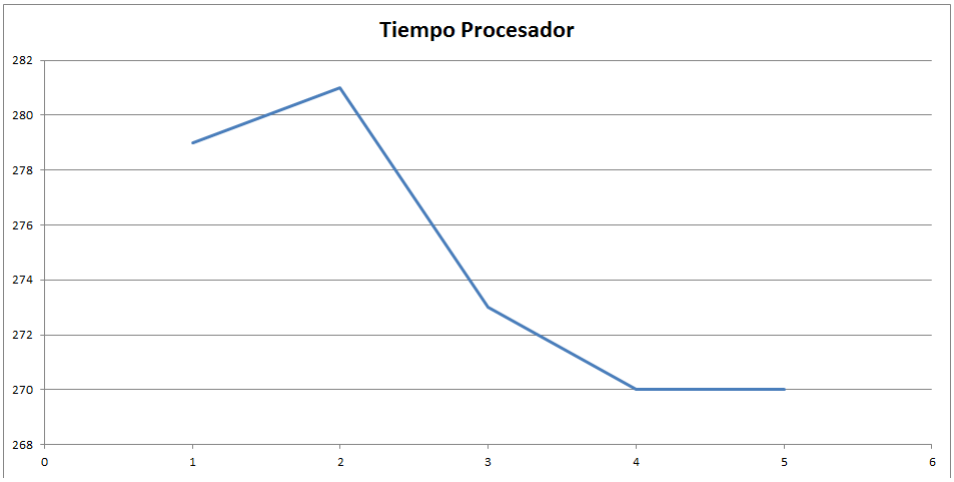

Figura 5.8: Benchmark para 100 Iteraciones Población 50

En la gráfica 5.9, se ejecutó el algoritmo varias veces, con un criterio de parada de 100 y un tamaño de población de 150 para cada una de las corridas.

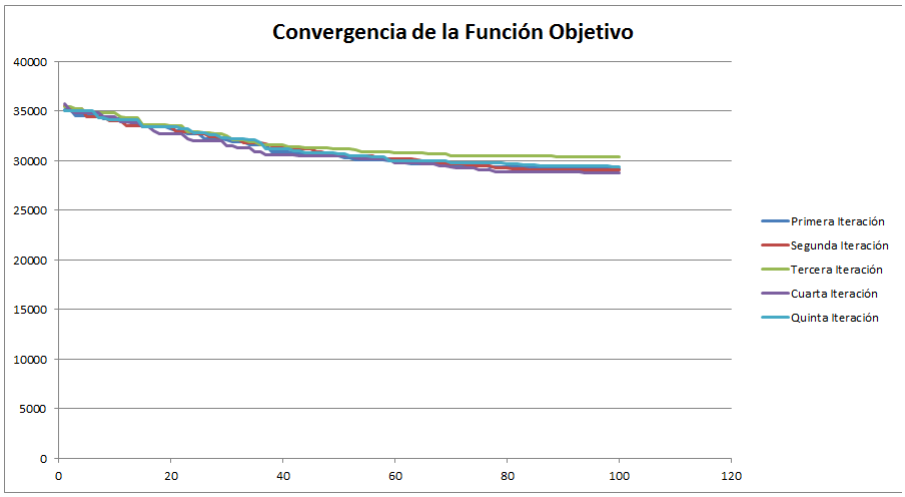

Figura 5.9: Criterio de parada de 100 Iteraciones, Población de 150

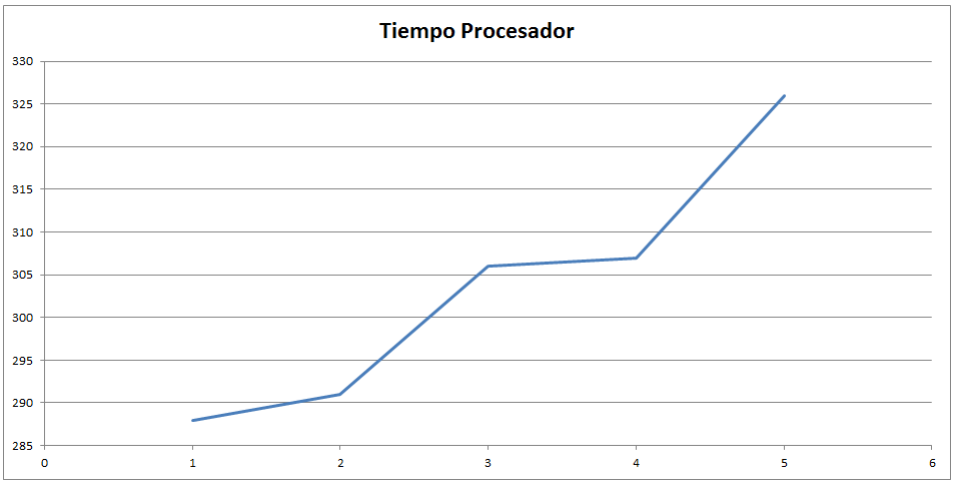

Figura 5.10: Benchmark para 100 Iteraciones Población 150

En la gráfica 5.11, se ha ejecutado el algoritmo 5 veces, con un criterio de parada de 100 iteraciones y un tamaño de población de 200 para cada una de las corridas.

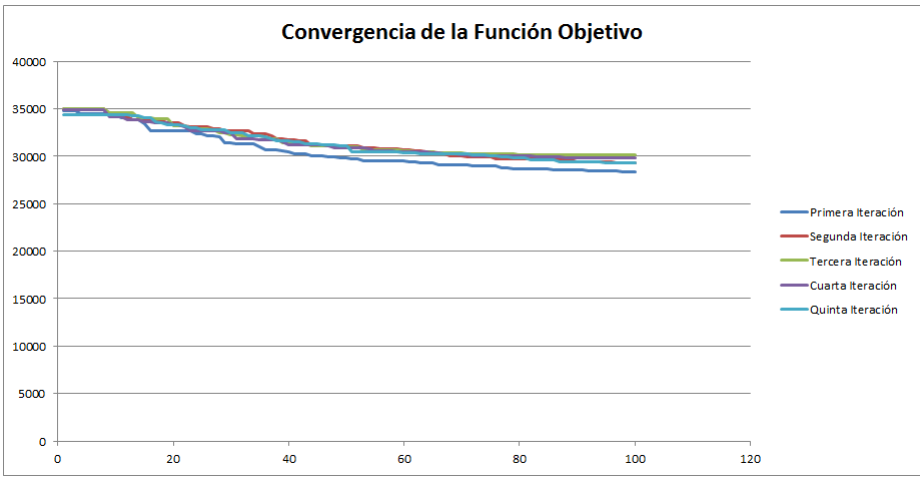

Figura 5.11: Criterio de parada de 100 Iteraciones, Población de 200

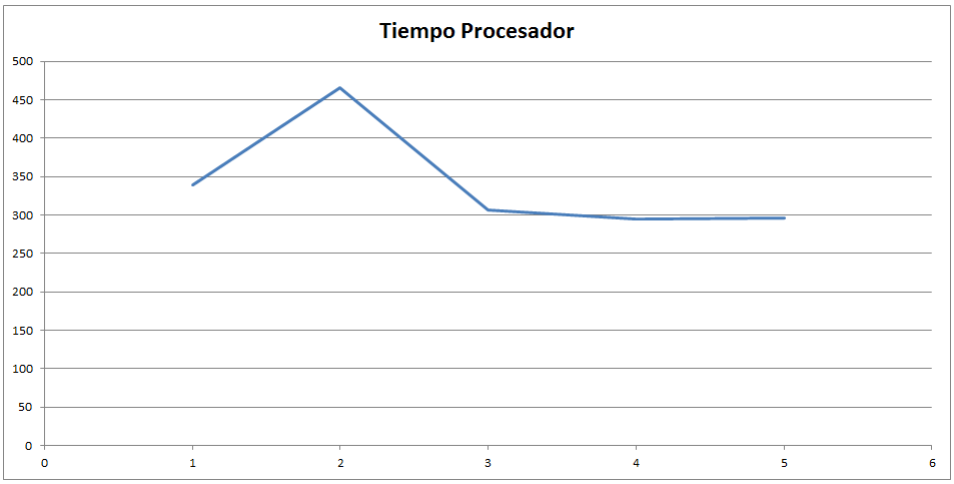

Figura 5.12: Benchmark para 100 Iteraciones Población 200

En la gráfica 5.13 tenemos un número de población elevado poco más elevado de 300, para poder determinar si en realidad el aumentar considerablemente el número de la población podemos llegar a valores de la función objetivo más pequeño.

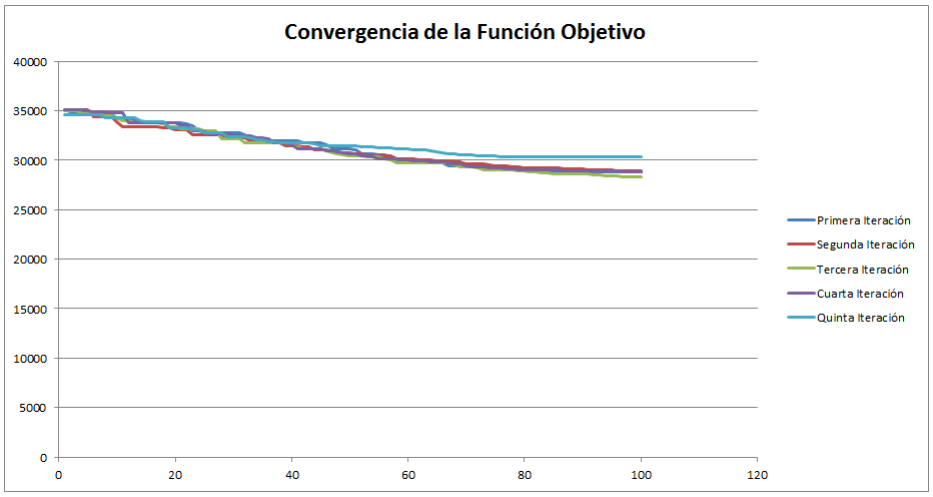

Figura 5.13: Criterio de parada de 200 Iteraciones, Población de 300

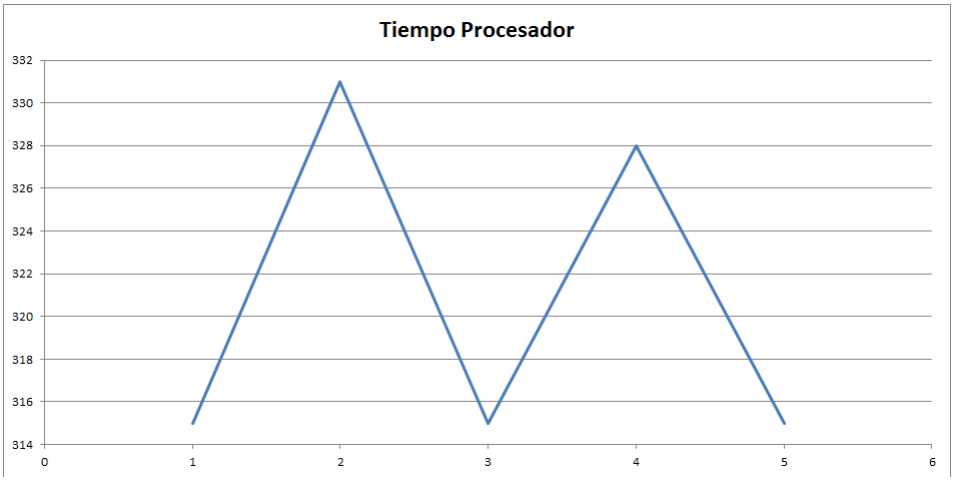

Figura 5.14: Benchmark para 100 Iteraciones Población 300

Finalmente, se puede concluir que con criterio de parada de 100 iteraciones y una población de 200, se puede obtener buenos resultados de la función objetivo.

5.2. Elección del cruce y eliminación

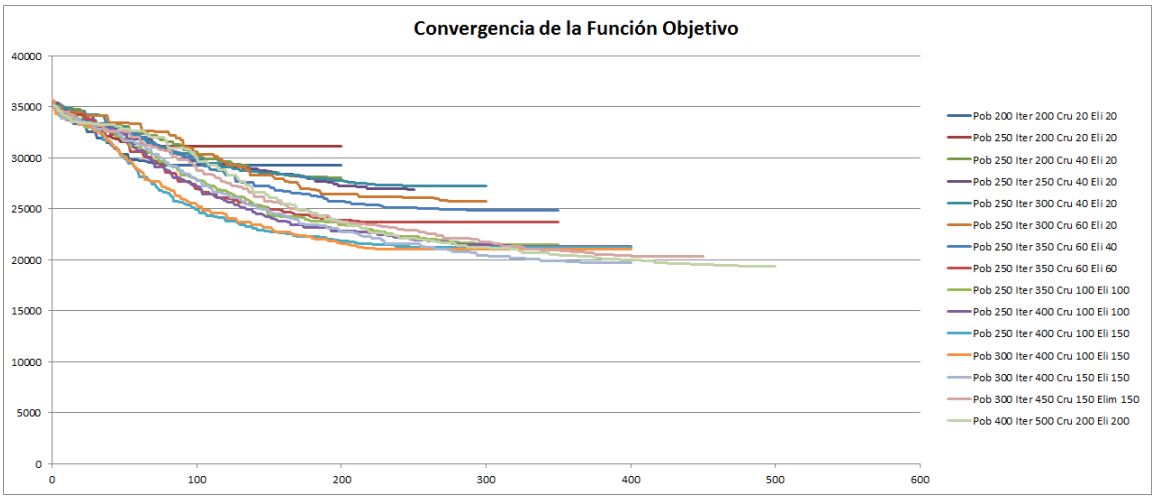

Figura 5.15: Gráfica Total

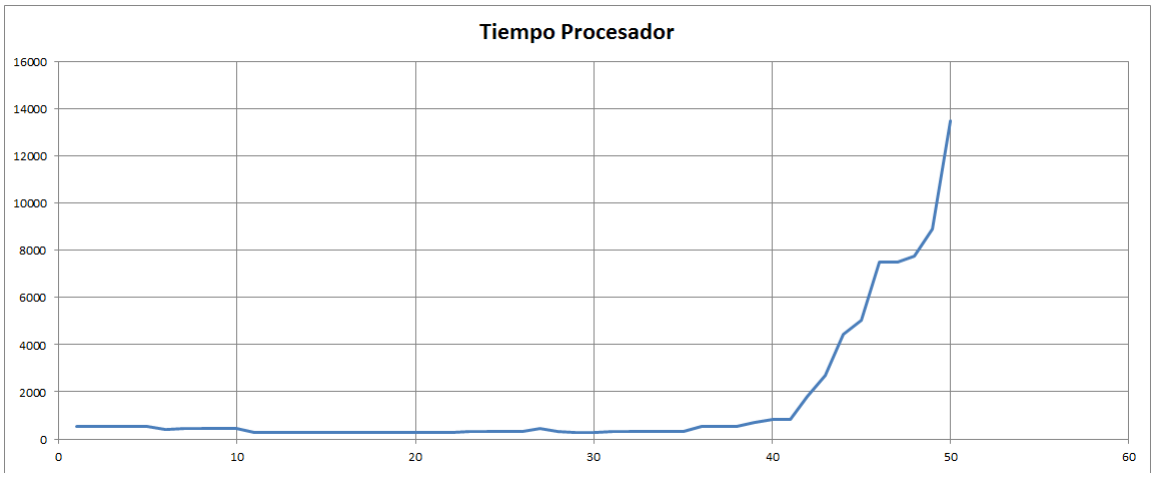

Figura 5.16: Benchmark

5.3. Resultados finales

Como se habla anteriormente el resultado que se obtiene después de la ejecución del algoritmo es un vector de colores al cual serán asociados todas las clases que se desean programar. Pero esto solamente sirve si antes tenemos el vector de clases que es el orden en el que tienen que ser asignadas todas las clases que deben ser programadas. Como

se muestra en la figura 5.17 $E = \{c_m, g_o, a_p, p_q, s_n\}$ y cada una nos dice exactamente qué es lo que se va a impartir, a que hora y con que profesor.

Vector de Colores = {15 34 39 14 40 32 13 44 37 29
30 3 12 33 10 7 39 5 34 35 27 13 23 1 44 24 17 42 20
21 45 39 19 42 15 13 45 19 22 44 4 23 37 3 18 45 14 2
38 13 21 40 11 41 31 13 39 18 3 42 34 7 36 12 22 41 4
17 38 16 14 14 44 22 45 1 20 23 43 21 16 23 41 18 12
13 38 7 16 34 15 5 39 15 39 19 1 44 20 14 38 24 22 23 21
2 10 28 16 21 42 23 32 8 28 41 3 29 30 4 17 37 36 8 35 16
10 38 30 31 10 36 6 13 32 37 12 38 22 40 16 36 34 15 41 43
14 11 34 6 32 24 31 19 23 41 17 13 42 15 20 39 11 8 20 41
12 18 40 19 42 38 16 43 18 20 41 26 6 45 21 2 3 37 36 6 30 38
11 31 28 40 16 6 41 17 19 42 5 20 17 15 43 27 6 36 8 7 37 34
11 32 31 6 37 29 1 39 19 29 11 40 29 10 28 9 34 26 7 31 8 5 40
28 10 33 29 11 8 30 3 6 36 7 29 8 26 18 7 36 31 10 37 33 9 32 28
5 27 7 6 29 4 24 43 2 30 27 25 20 26 29 5 24 43 3 26 44 4 25 8 31
35 2 14 32 12 30 42 19 15 26 25 3 43 45 1 22 17 14 18 39 26 24 25
1 18 29 17 36 35 17 27 4 26 37 12 30 25 27 4 7 31 5 23 30 2 28 32 25
6 11 32 25 44 25 2 26 32 7 27 45 4 33 35 19 42 11 12 45 1 27 33 3 22
34 9 33 9 35 15 27 9 28 8 12 35 15 33 10 33 21 39 9 35 16 28 18 22 42
20 21 1 19 40 44 20 1 40 21 22 41 14 9 2 43 22 4 44 24 1 38 25 28 24
2 26 23 4 43 31 5 18 37 24 5 44 13 9 45 17 25 45 3 24 43 2 23 35 5 34
9 33 13 9 38 14 8 40 16 12 35 10 33 10 30 11}

Figura 5.17: Vector de Colores

Después se obtiene el vector resultado el cual se ordena según [16] donde toma el concepto de método de la llave aleatoria, esta técnica puede ser aplicada de manera muy simple, cualquier ordenamiento puede ser usando el método de quick sort debido al tiempo computacional que implica, *O(nlogn)*. Después de obtener la secuencia, que en este caso será el vector solución, se aplica el método de llave aleatoria para ordenar los valores en orden ascendente.

El cual da un valor de función objetivo de:

Poblacion: 300
Iteraciones: 400
Pobacion a Reemplazar: 150
Poblacion a Cruzar: 150
Colores Totales: 45
Distancia Minima: 18
Clases por Color: 10
Benchmark: 7770 wallclock secs (7758.55 usr + 0.42 sys = 7758.97 CPU)

Tabla 5.1: Resultados computacionales de la buena solución

Ahora se tiene el vector de colores, sin embargo todavía hay que darle un tratamiento para poder entender los datos fácilmente.

Ordenando los datos se obtiene

					LUNES					
1	PO_{21}	MMII_{22}	GGIA_{12}	PEECM_{23}	FI_{23}	SC_{22}	HAG_{31}	SIGII_{21}	APC_{31}	CRO_{13}
2	PCE_{12}	HMT_{12}	MS_{33}	TAR_{12}	TII_{12}	TIII_{33}	GDIA_{21}	CRO_{11}	EG_{31}	HDFIA_{32}
3	EE_{31}	MMI_{33}	HB_{32}	IS_{32}	ST_{11}	HU_{32}	TI_{22}	FI_{22}	HAG_{32}	HDFIA_{31}
4	MMI_{32}	IAMRM_{22}	IS_{33}	TAR_{11}	TI_{23}	MM_{22}	TIII_{31}	GDIA_{23}	CRO_{12}	EG_{32}
5	EE_{33}	AE_{23}	APV_{23}	CMT_{32}	TACH_{22}	TI_{21}	TIII_{32}	EG_{33}	GHM_{12}	HDFIA_{33}
6	DE_{12}	MT_{12}	MS_{32}	ST_{12}	APV_{31}	PEM_{22}	PEECM_{22}	HU_{13}	TACH_{23}	AP_{22}
7	EE_{13}	HB_{33}	AE_{12}	PEM_{23}	CMT_{31}	HU_{33}	OH_{22}	TACH_{13}	TIII_{12}	GDIA_{22}
8	IS_{31}	CS_{12}	AGPA_{33}	PEM_{13}	CMT_{22}	HU_{12}	OH_{21}	TII_{11}	PRI_{22}	PGM_{12}
9	IP_{22}	TACH_{21}	HAG_{33}	TEII_{21}	PRI_{21}	PRII_{11}	APC_{33}	GHM_{13}	LT_{21}	PGM_{11}
					MARTES					
10	EE_{32}	HTM_{22}	CS_{13}	DE_{21}	IP_{21}	CMT_{33}	OH_{23}	SIGI_{11}	PGM_{33}	TYM_{21}
11	PCE_{33}	MT_{11}	AGPA_{23}	ST_{13}	PEECM_{21}	AI_{12}	HU_{31}	CGA_{11}	GG_{22}	TYM_{22}
12	EE_{12}	IAMRM_{21}	AE_{11}	DE_{23}	HDIA_{31}	TII_{13}	TEI_{12}	HAG_{11}	GPS_{11}	PGM_{13}
13	GTIA_{13}	EP_{22}	MMI_{11}	PCE_{32}	HB_{31}	AE_{21}	DE_{22}	AGPA_{31}	GHM_{32}	LT_{22}
14	GTIA_{12}	PCE_{31}	IAMRM_{33}	MMII_{11}	GGIA_{13}	HC_{33}	TII_{22}	FII_{12}	APC_{23}	PGM_{31}
15	GTIA_{11}	FMMC_{22}	AE_{13}	GGIA_{11}	HC_{32}	AGPA_{22}	PEM_{21}	FI_{21}	TEII_{22}	GPS_{12}
16	IAMRM_{23}	TGS_{11}	AE_{22}	HMT_{13}	CS_{32}	HC_{31}	HDIA_{33}	APV_{21}	PRII_{12}	PGM_{32}
17	PO_{22}	IAMRM_{32}	CS_{11}	AGPA_{21}	APV_{22}	PEM_{11}	FII_{11}	FG_{12}	MM_{21}	GHM_{33}
18	PCE_{11}	HB_{22}	TGS_{22}	HDIA_{12}	MS_{21}	OH_{12}	FII_{12}	FG_{11}	PG_{11}	GHM_{11}
					MIERCOLES					
19	FMMC_{21}	MMI_{31}	GGIA_{31}	P_{12}	HDIA_{32}	APV_{32}	AI_{11}	FI_{11}	GG_{21}	SIGII_{12}
20	PO_{13}	MMII_{32}	GGIA_{32}	AGPA_{32}	HDIA_{11}	MS_{31}	APV_{33}	TAR_{23}	PG_{22}	APC_{21}
21	PO_{23}	PCE_{13}	MMII_{33}	HTM_{31}	HMT_{23}	MS_{23}	SIGI_{12}	SIGII_{11}	APC_{22}	
22	MMI_{12}	IAMRM_{31}	MMII_{31}	HMT_{11}	HC_{11}	FI_{33}	HAG_{13}	PG_{21}	APC_{32}	CRO_{31}
23	PO_{11}	MMI_{13}	MMII_{13}	TGS_{21}	HMT_{21}	IS_{11}	P_{22}	TIII_{13}	EG_{22}	HDFIA_{13}
24	PO_{12}	GGIA_{33}	P_{11}	TAR_{21}	TI_{31}	SC_{21}	CRO_{32}	EG_{21}	GHM_{31}	HDFIA_{12}
25	TAR_{13}	TI_{33}	FI_{12}	SC_{12}	TIII_{11}	AP_{12}	CGA_{12}	GDIA_{11}	CRO_{33}	HDFIA_{11}
26	MS_{22}	CMT_{21}	OH_{31}	TAR_{33}	TI_{32}	FI_{31}	SC_{11}	TEI_{11}	GDIA_{31}	EG_{12}
27	EP_{12}	PEM_{12}	TACH_{32}	TAR_{32}	MM_{12}	TIII_{21}	GDIA_{32}	HAG_{12}	PRI_{11}	
					JUEVES					
28	HTM_{32}	IS_{12}	ST_{33}	IP_{12}	CMT_{23}	TACH_{12}	AP_{11}	PRI_{12}	PRII_{22}	EG_{11}
29	EE_{11}	IS_{13}	PEECM_{13}	AI_{21}	IP_{11}	HU_{21}	OH_{11}	TACH_{33}	TI_{11}	FG_{21}
30	EE_{21}	IS_{23}	CS_{33}	ST_{22}	HU_{22}	TAR_{22}	TII_{23}	TEI_{22}	TIII_{23}	TYM_{12}
31	HB_{21}	DE_{11}	P_{21}	ST_{23}	PEECM_{12}	CMT_{12}	OH_{13}	TII_{21}	TIII_{22}	EG_{23}
32	GTIA_{32}	IS_{21}	DE_{32}	MT_{22}	PEECM_{31}	TACH_{31}	TII_{32}	AP_{21}	CGA_{21}	GDIA_{12}
33	EE_{22}	HU_{11}	TACH_{11}	GDIA_{33}	HAG_{22}	TEII_{11}	GPS_{22}	SIGI_{21}	LT_{12}	TYM_{11}
34	GTIA_{21}	EP_{11}	HB_{23}	AE_{32}	HC_{22}	MT_{21}	PEECM_{11}	CMT_{11}	HAG_{23}	LT_{11}
35	EE_{21}	CS_{22}	TII_{31}	MM_{11}	GG_{11}	TEII_{12}	GPS_{21}	PRII_{21}	HDFIA_{23}	PGM_{23}
36	IAMRM_{11}	CS_{31}	DE_{31}	HC_{12}	ST_{31}	PEM_{32}	HU_{23}	OH_{32}	FG_{22}	
					VIERNES					
37	GTIA_{33}	MMI_{23}	CS_{21}	DE_{13}	ST_{21}	PEM_{33}	PEECM_{32}	OH_{33}	TEI_{21}	GHM_{21}
38	PCE_{22}	IAMRM_{13}	AE_{31}	GGIA_{23}	CS_{32}	DE_{33}	HDIA_{23}	ST_{32}	CRO_{23}	PGM_{21}
39	GTIA_{31}	EE_{23}	FMMC_{11}	HB_{12}	AE_{33}	GGIA_{21}	AGPA_{13}	PEECM_{33}	FII_{22}	SIGI_{22}
40	GTIA_{22}	PCE_{23}	HC_{21}	HDIA_{22}	APV_{11}	AI_{22}	CMT_{13}	SIGII_{22}	APC_{12}	PGM_{22}
41	HB_{11}	IAMRM_{12}	TGS_{12}	IS_{22}	HC_{13}	AGPA_{11}	HDIA_{21}	MS_{12}	APV_{12}	APC_{13}
42	PO_{32}	FMMC_{12}	HB_{13}	HMT_{33}	AGPA_{12}	HDIA_{13}	APV_{13}	TII_{33}	GG_{12}	PG_{12}
43	MMII_{23}	HC_{23}	MS_{11}	PEM_{31}	TAR_{31}	TI_{12}	FI_{32}	CRO_{21}	EG_{13}	HDFIA_{22}
44	GTIA_{23}	PO_{31}	MMI_{22}	MMII_{21}	GGIA_{22}	TI_{13}	CGA_{22}	APC_{11}	CRO_{22}	GHM_{22}
45	PO_{33}	MMI_{21}	PCE_{21}	MMII_{12}	MS_{13}	FI_{13}	GDIA_{13}	HAG_{21}	GHM_{23}	HDFIA_{21}

Figura 5.18: Resultados Ordenados

Y se puede observar que las clases son calendarizadas de acuerdo a la forma que tiene de programarlas la División de Ingenierias Civil y Geomática.

5.4. Comparativa con una programación tradicional

Se realizó la impresión de los horarios obtenidos mediante el algoritmo propuesto y se comparó con los que se tenían ya programados para el semestre en curso, y también de semestres anteriores, y como resultado se observó que con un mejor acomodo se eliminó totalmente los grupos sin salón y también el que se programaran dos clases en el mismo

horario y mismo salón, se demostró también que se puede tener una mayor ocupación de los salones, a diferencia de la programación tradicional, en donde, algunos salones no tenían programada ninguna clase en alguno de los horarios posibles o incluso varias horas seguidas.

Los inconvenientes son que, en algunos casos, los profesores tienen predilección por cierto horario en cierto salón, y debido a que son factores humanos que cambian semestre con semestre, no se puede adelantar la programación mediante el algoritmo propuesto, sin embargo, estas preferencias pueden ser fácilmente programadas e incluidas en la función objetivo, pero esto se deja para trabajos futuros.

Capítulo 6

Conclusiones

Este trabajo presenta un algoritmo heurístico para efectuar la programación de horarios y asignación de salones mediante el uso de Algoritmos Genéticos para la División de Ingenierias Civil y Geomática de la Facultad de Ingeniería de la Universidad Nacional Autónoma de México. Este algoritmo asigna simultáneamente todas las clases con los respectivos horarios y salón de las asignaturas, este enfoque se basa en lo estudiado por [5], [2], [38], [35] y que se aplicó de manera real a la DICyG, además se diferencia de otros modelos por el número de penalizaciones y restricciones que se manejan.

Considerando este tamaño de problema, la tarea de generar la programación de las clases manualmente se transforma en una labor en extremo compleja y sujeta múltiples errores como los presentados en el apartado 2.2. La utilización de esta nueva herramienta propuesta garantiza la satisfacción de todos los requerimientos obligatorios (como no permitir la impartición simultanea de dos clases del mismo grupo y misma asignatura) y mejora en foma significativa el cumplimiento de las condiciones deseables. Además, se reduce sustancialmente el tiempo requerido para la obtención de la programación de horarios y asignación de salones de clase.

Actualmente, se ha puesto a disposición de las autoridades y del personal encargado de la asignación de salones para la DICyG, el algoritmo propuesto y se ha abierto la puerta para posibles mejoras que se pudieran encontrar con el uso constante del algoritmo, esto es, que aunque se han considerado la mayoría de las restricciones y lo deseable de los horarios, quedando a posibles modificaciones, además debido a la sencillez con la que esta programado se pueden ir incorporando nuevas restricciones con el fin de que sea mucho más fácil y en un solo paso dicha asignación.

Con la implementación del sistema computacional será posible automatizar el proceso de generación de la programación de horarios y asignación de salones aumentando la calidad de las programaciones, la flexibilidad ante cambios en los requerimientos y condiciones deseables y la posibilidad de explorar múltiples escenarios. Esto es, si en dado caso se llegara a aumentar el número de salones o si dichos salones tienen la peculiaridad de tener equipos de cómputo, entonces se pueden ir aumentando la preferencia por que estos nuevos salones se asignen a asignaturas cuya necesidad de utilizar equipo de cómputo sea mayor a la de una impartición totalmente teórica.

Una línea interesante para profundizar en el trabajo de investigación es la adaptación del algoritmo propuesto para algoritmos evolutivos y sistemas expertos, en donde se pretende que sea la base para el desarrollo de un sistema aún más complejo para la programación de horarios, por ejemplo, añadiendo además redes neuronales para que el sistema vaya aprendiendo semestre con semestre.

Bibliografía

[1] M. Aboytes-Ojeda. Algoritmo de búsqueda tabú para una variante del problema de coloración, master thesis. *Universidad Nacional Autónoma de México*, page 61, 2011.

[2] D. Abramson and J. Abela. Algoritmos genéticos. *Australian Computer Science Conference*, pages 1–11, 1992.

[3] D. Abramson and J. Abela. A parallel genetic algorithm for solving the school timetabling problem. In *Proceedings of the Fifteenth Australian Computer Science Conference*, volume 14, pages 1–11, 1992.

[4] R. Álvarez-Valdés, E. Crespo, and J. M. Tamarit. A tabu search algorithm to schedule university examinations. *Questiio*, 21(1):201, 1997.

[5] L. Araujo and C. Cervigón. *Algoritmos Evolutivos, Un enfoque práctico.* Alfaomega Grupo Editor, RA-MA, 2009.

[6] Z. N. Azimi. Hybrid heuristic for examination timetabling problem. *Applied Mathematics and Computation*, 163:705 – 733, 2005.

[7] K. R. Baker and D. Trietsch. *Principles of Sequencing and Scheduling.* John Wiley and Sons, 2009.

[8] P. Boizumault, C. Guéret, and N. Jussien. Efficient labelling and constraint relaxation for solving time tabling problems. *Constraint Languages and their use in Problem Modelling International Logic Programming Symposium-Workshop*, 1994.

[9] E. K. Burke, D. G. Elliman, and R. F. Weare. The automation of the timetabling process in higher education. *Journal of Educational Technology Systems*, 23:353 – 359, 1995.

[10] E. K. Burke, K. Jackson, J. H. Kingston, and R. Weare. Automated university timetabling: The state of the art. *The Computer Journal*, 40(9):565–571, Enero 1997.

[11] E. K. Burke and J. Newall. A multi-stage evolutionary algorithm for the timetable problem. *IEEE Computational Intelligence Society*, 3:63–74, 1999.

[12] E. K. Burke, S. Petrovic, and Q. Rong. Case-based heuristic selection for timetabling problems. *Journal of Scheduling*, 9:115–132, 2006.

[13] E. K. Burke and P. S. Recent research directions in automated timetabling. *European Journal Of Operational Research*, 140:266–280, 2002.

[14] M. Cangalovic and J. A. Schreuder. Exact colouring algorithm for weighted graphs applied to timetabling problems with lectures of different lengths. *European Journal of Operational Research*, 51(2):248–258, 1991.

[15] M. W. Carter. A survey of practical applications of examination timetabling algorithms. *Operations Research*, 34(2):193–202, 1986.

[16] S.-H. Chen, P.-C. Chang, C.-L. Chan, and M. V. A hybrid electromagnetism-like algorithm for single machine scheduling problem. *Expert Systems with Applications*, 36(2, Part 1):1259 – 1267, 2009.

[17] A. Cortéz. Teoría de la complejidad computacional y teoría de la computabilidad. *Revista de investigación de sistémas e informática*, 1:102–105, 2004.

[18] L. Cruz-Reyes, F. Alonso-Pecina, and L. A. Ramírez-Gómez. *El problema de la Programación de Horarios con Coloreo de Grafos.* PhD thesis, Instituto Tecnológico de la Ciudad de Madero, 2002.

[19] S. de-los Cobos, J. Goddard, M. A. Gutiérrez, and A. Martínez. *Búsqueda y exploración estocástica.* Universidad Autónoma Metropolitana, 2010.

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

[21] D. de Werra. The combinatorics of timetabling. *European Journal of Operational Research*, 96:504–513, 1997.

[22] A. Duarte, J. J. Pantrigo, and M. Gallego. *Metaheurísticas*. Librería-Editorial Dykinson, 2007.

[23] S. Even, A. Itai, and S. A. On the complexity of time table and multi-commodity flow problems. *SIAM Journal on Computing*, pages 691–703, 1976.

[24] F. Glover and G. A. Kochenberger. *Handbook of metaheuristics*. Springer, 2003.

[25] C. Head and S. Sami. A heuristic approach to simultaneous course/student timetabling. *Computer and Operations Research*, 34:919–933, 2007.

[26] R. Hernández, J. P. Miranda, and P. A. Rey. Programación de horarios de clases y asignación de salas en la facultad de ingeniería de la universidad diego portales. *Revista Ingeniería de Sistemas*, XXII, 2008.

[27] J. H. Holland. Algoritmos genéticos. *Investigación y Ciencia*, 192:38–45, 1992.

[28] P. Lara-Velázquez, R. López-Bracho, J. Ramírez-Rodríguez, and J. Yánez. A model for timetabling problems with period spread constraints. *Journal of the Operational Research Society*, 62:217–222, 2011.

[29] N. G. Londono. Algoritmos genéticos. *Univerisidad Nacional de Colombia, Escuela de Estadística, Sede Medellín*, page 65, 2006.

[30] M. Nandhini and S. Kanmani. A survey of simulated annealing methodology for university course timetabling. *International Journal of Recent Trends in Engineering*, 1(2):255–257, 2009.

[31] G. A. Neufeld and J. Tartar. Graph coloring conditions for the existence of solutions to the timetable problem. *Commun. ACM*, pages 450–453, 1974.

[32] C. H. Papadimitriou. Computational complexity. In *Encyclopedia of Computer Science*, pages 260–265. John Wiley and Sons Ltd.

[33] C. Pérez-de-la Cruz and J. Ramírez-Rodríguez. Un algoritmo genético para un problema de horarios con restricciones especiales. *Revista de Matemática: Teoría y Aplicaciones, 18(2)*, pages 217–232, 2010.

[34] M. Pinedo and X. Chao. *Operations Scheduling with applications in manufacturing and services*. Irwin McGraw-Hill, 1999.

[35] J. Ramírez-Rodríguez. *Extensiones del problema de coloración de grafos*. PhD thesis, Universidad Complutense de Madrid, 2001. http://www.ucm.es/BUCM/tesis/mat/ucm-t24770.pdf.

[36] J. Ramírez-Rodríguez, M. A. Gutiérrez, P. Lara, and R. López. Un algoritmo evolutivo para resolver el problema de coloración robusta. *Revista de Matemática: Teoría y Aplicaciones*, pages 111–120, 2004. http://simmac.emate.ucr.ac.cr/index.php/revista/article/viewFile/165/149.

[37] S. Rayward, I. Osman, C. Reeves, and G. Smith. *Modern heuristic search methods*. John Wiley and Sons, Chichester, England, 1996.

[38] P. Ross and D. Corne. Comparing genetic algorithms, simulated annealing, and stochastic hillclimbing on timetabling problems. *Springer Berlin / Heidelberg*, 993:94–102, 1995.

[39] O. Rossi-Doria, M. Sampels, M. Birattari, M. Chiarandini, M. Dorigo, L. M. Gambardella, J. Knowles, M. Manfrin, M. Mastrolilli, and B. Paechter. A comparison of the performance of different metaheuristics on the timetabling problem. *Springer Berlin / Heidelberg*, 2740:329–351, 2003.

[40] M. G. Tallavó and A. A. Martínez. Algoritmo basado en tabu search para el problema de asignación de horarios de clases. *Faraute de Ciencia y Tecnología*, 1(1), Julio 2006. Venezuela.

[41] J. Yánez and J. Ramírez-Rodríguez. The robust coloring problem. *European Journal of Operational Research*, pages 546–558, 2003.

Anexo A

Concentrado de materias de la División de Ingenierías Civil y Geomática

Asignaturas que se imparten en la División de Ingenierías Civil y Geomática

	No. de horas Diarias				Semanal
	1.5	2.25	3	4.5	
Administración en Ingeniería	4		1		3
Construcción I				1	4.5
Construcción II	1				3
Construcción III			1		3
Movimiento de Tierras	4		1		3
Presupuestación de Obras	4	4		1	4.5
Programación y Construcción de Estructuras	3	5			4.5
Análisis Estructural	8	1			4.5
Diseño Estructural	6	1			4.5
Estatica Estructural	8	3			4.5
Estructuras Isostaticas	1				4.5
Mecánica de Materiales I	9				4.5
Mecánica de Materiales II	7	1			4.5
Proy. Estruct. P/Edific. Conc y Mamp.	1	1			4.5
Proyecto de Estructuras Metálicas	1	1			4.5
Temas Especiales de Ing. Civil I	1	2			3
Temas Especiales de Ing. Civil II	2	2			4.5
Cimentaciones	3	1			4.5
Comportamiento de Suelos	3	2			4.5
Fundamentos de mecánica del medio continuo	3				3
Geología	2	3			4.5
Mecánica de Suelos	3	2			4.5
Mecánica del medio continuo	1				3
Hidráulica Básica	8				4.5
Hidráulica de Canales	6				4.5
Hidráulica de Máquinas y Transitorio	6				4.5
Hidráulica Urbana	1	1			4.5
Hidrología	4	1			4.5
Obras Hidráulicas	3				4.5
Temas Especiales de Ing. Civil III	2	1			4.5
Abastecimiento de agua potable y alcantarillado	2	3			4.5
Impacto ambiental	2				4.5
Impacto ambiental y manejo de residuos municipales	5				4.5
Tratamiento de aguas para consumo humano	1				4.5
Tratamiento de aguas residuales	4	1			4.5
Evaluación de proyectos	4				4.5
Ingeniería de Sistemas	5	2			4.5
Ingeniería de Sistemas I	1				3
Ingeniería de Sistemas II	1				4.5
Integración de proyectos	4		2		3
Planeación	6				3

Tabla 6.1: Concentrado de Materias de la DICyG parte 1

Asignaturas que se imparten en la División de Ingenierías Civil y Geomática

	No. de horas Diarias				Semanal
	1.5	2.25	3	4.5	
Sistemas de Transporte	4	2			4.5
Teoría general de sistemas	5				3
Administración de proyectos	2				3
Catastro		2			4.5
Geomática	3	5		2	4.5
Hidrografía	2				4.5
Hidrología aplicada a la geomática		1			4.5
Hidrología básica		1			4.5
Legislación topográfica	1				4.5
Obras de Infraestructura		1			4.5
Prácticas generales				1	4.5
Principios de Administración	1				3
Topografía		1			4.5
Topografía de yacimientos minerales	2				3
Topografía general y prácticas				1	4.5
Topografía I	4			4	4.5
Topografía II	3			3	4.5
Topografía III		1		1	4.5
Cartografía	1		1		3
Fundamentos de geodesia		2	2		4.5
Geodesia	1	1			4.5
Geología y Geomorfología	1		1		3
Geomática para ciencias de la tierra		1		4	4.5
Geomorfología	1				3
Modelación matemática	1				3
Sistemas de posicionamiento Global	2	2			3
Sistemas de Coordenadas	2				3
Teoría de los Errores I	1				3
Teoría de los Errores II	1				3
Bienes raíces y avalúos	1				3
Fotogrametría I	4		4		3
Fotogrametría II	3	2			4.5
Percepción remota			2		6
Percepción remota I			2		6
Percepción remota II			1		6
Proyecto Geomático	1				3
Sistemas de Información Geográfica I	1		7		3
Sistemas de Información Geográfica II	2		2		3

Tabla 6.2: Concentrado de Materias de la DICyG parte 2

Anexo B

Plan de Estudios de la carrera de Ingeniería Civil impartida en la Facultad de Ingeniería de la Universidad Nacional Autónoma de México campus Ciudad Universitaria

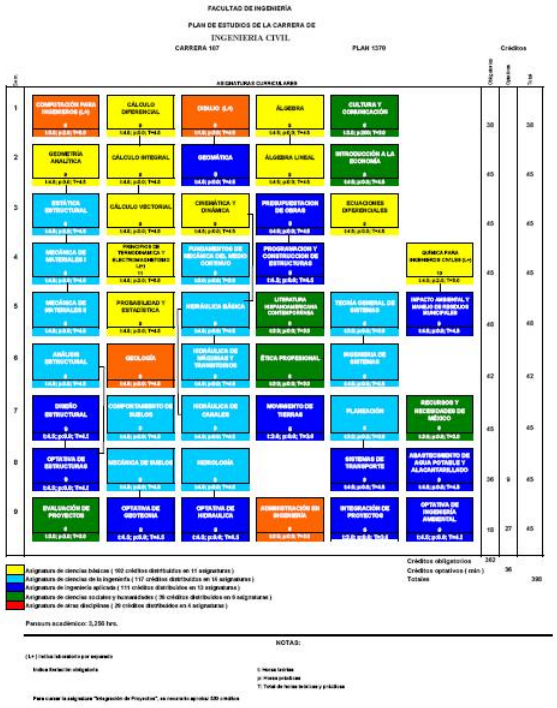

Figura 6.1: Plan de Estudios de la carrera de Ingeniería Civil

Plan de estudios de la carrera de Ingeniería Geomática impartida en la Facultad de Ingeniería de la Universidad Nacional Autónoma de México campus Ciudad Universitaria

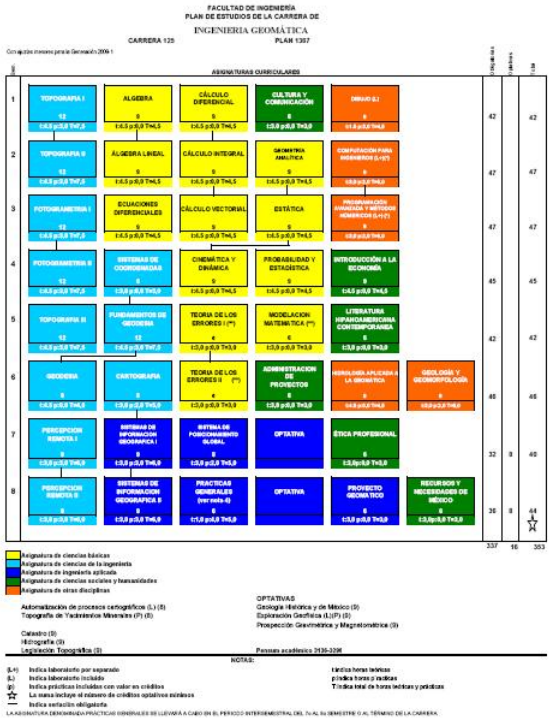

Figura 6.2: Plan de Estudios de la carrera de Ingenieria Geomática

Anexo C

Código Fuente

```
my $start_time = new Benchmark;
my $end_time = new Benchmark;
my $difference = timediff($end_time, $start_time);

print "\n\n Poblacion: " . POBLACION;
print "\n Iteraciones: " . ITERACIONES;
print "\n Pobacion a Reemplazar: " . P_REEMPLAZAR;
print "\n Poblacion a Cruzar: " . P_CRUZAR;
print "\n Colores Totales: " . COLORES;
print "\n Distancia Minima: " . DISTANCIA_MINIMA;
print "\n Clases por Color: " . CLASES_POR_COLOR;
print "\n Benchmark: ", timestr($difference), "\n\n";
sub itera {
        my $poblacion=POBLACION;
        my $iteraciones=ITERACIONES;
        my $p_reemplazar=P_REEMPLAZAR;

        my $p_cruzar=P_CRUZAR;
        my $obj=ColorizacionHorarios::new();


        $obj->{Colores_Totales}=COLORES;
        $obj->{Distancia_Minima}=DISTANCIA_MINIMA;
        $obj->{Clases_Por_Color}=CLASES_POR_COLOR;

        $obj->Obtener_Matriz_de_Datos();
        $obj->Obtener_Restricciones();
        $obj->Obtener_Penalizaciones();
        $obj->Imprimir_Matriz_Restricciones_Penalizaciones();
        $obj->Imprimir_Matriz_Penalizaciones_Por_Distancia();
        for(my $i=0; $i<$poblacion; $i++){
                $obj->Generar_Individuo_Aleatorio();
        }
        $obj->Evaluar_Poblacion();
        $obj->Ordenar_Poblacion();
        $obj->Imprimir_Poblacion();
```

```
        for(my $i=1; $i<$iteraciones;$i++){

                $obj->Evoluciona_Poblacion($p_reemplazar,$p_cruzar);
                $obj->Evaluar_Poblacion();
                $obj->Ordenar_Poblacion();
                print "\n";
                $obj->Imprimir_Poblacion();
                print "\n";
        }
}

package ColorizacionHorarios;

use constant DEBUG => 0;
use strict;
sub new {
        my $obj={};
        bless $obj, "ColorizacionHorarios";

        $obj->{Colores_Totales}=45;
        $obj->{Clases_Por_Color}=11;
        $obj->{Distancia_Minima}=18;
        $obj->{Penalizacion_Distancia_Minima}=50;
        $obj->{Penalizacion_Diferente_Materia}=1;
        $obj->{Penalizacion_Misma_Clase_Diferente_Grupo}=50;
        $obj->{Penalizacion_Diferente_Clase_Mismo_Grupo}=100;

        $obj->{Evaluado}=0;
        $obj->{Ordenado}=0;
        return $obj;
}

sub Evoluciona_Poblacion {
                my $obj=shift;
                die "Individuos no generados\n"
                        unless defined $obj->{Poblacion};
                die "No se puede evolucionar Poblacion por no estar ordenada\n"
                        unless $obj->{Ordenado} == 1;

                my $n_eliminados=shift || int(@{$obj->{Poblacion}} * 0.2);

                my $n_a_cruzar=shift || $n_eliminados;
                die "No se pueden cruzar $n_a_cruzar elementos"
                        if $n_a_cruzar <= 1;
                die "No se pueden eliminar $n_eliminados elementos"
                        if $n_a_cruzar < 1;

                for (my $i=0; $i<$n_eliminados; $i++){
                        pop @{$obj->{Poblacion}};
                }
```

```
                for (my $i=0; $i<$n_eliminados; $i++){
                        $a=int(rand($n_a_cruzar));
                        do{
                                $b=int(rand($n_a_cruzar));
                        } while ($a == $b);
                        $obj->Cruzar_Individuos($a,$b);
                }


                $obj->{Evaluado}=0;
        $obj->{Ordenado}=0;
        return $obj->{Poblacion};
}

Cruzar_Individuos {
                my $obj=shift;

                my ($p,$m)=@_;
                die "Numero de individuo padre a cruzar no definido\n"
                        unless defined $p;
                die "Numero de individuo madre a cruzar no definido\n"
                        unless defined $m;
                die "Individuos no generados\n"
                        unless defined $obj->{Poblacion};
                die "Matriz de Restricciones y Penalizaciones no Definida\n"
                        unless defined $obj->{M_Restricciones_Penalizaciones};

                my $Mrp=$obj->{M_Restricciones_Penalizaciones};
                my $Max=$obj->{Clases_Por_Color};

                my @ContC;
                my $individuo_eliminado=0;

                my $padre=$obj->{Poblacion}[$p]{Colores};
                my $madre=$obj->{Poblacion}[$m]{Colores};

                my $N=@$padre;
                my @mascara;

INICIA_CRUCE:
                my $hijo;

                for (my $i=0; $i<$N; $i++){
                        $mascara[$i]=int(rand(2));
                }
                if (DEBUG == 1) {
                                                print "\n-----Mascara de cruce entre padre $p y madre $m-
                                                print "$_ " foreach @mascara;
                                                print "\n-------Finaliza Mascara de Cruce-------\n\n";
                }
                for(my $i=0; $i<=$obj->{Colores_Totales};
                        $i++)
```

```
                {
                        $ContC[$i]=0;
                }
for(my $i=0;

                        $i<$N; $i++)
                {
                $color_invalido=0;
                my $intento=0;
                $color_invalido=0;
        my $padre_flag=0;
        if($mascara[$i]==1)
                                                {
        $hijo->[$i]=$padre->[$i];
        $padre_flag=1;
                                                }

else
        {
                $hijo->[$i]=$madre->[$i];
        }

        if($ContC[$hijo->[$i]] >= $Max )
                {
                $color_invalido=1;
                print "Color $hijo->[$i] del padre invalido en clase $i por exceder maxi
        if DEBUG == 1;
        }
else
                {
for(my $j=0; $j<$i; $j++)
                {
if (defined $Mrp->[$j][$i] && $Mrp->[$j][$i]  == 1 && $hijo->[$j] == $hijo->[$i]){
                                print "Color $hijo->[$i] del padre invalido en clase $i
if DEBUG == 1;
                        }                                       }                       }
        if($color_invalido==1)
                {
                        $color_invalido=0;

        if($padre_flag==1)
                {
                        $hijo->[$i]=$madre->[$i];
                }
                        else {
                                        $hijo->[$i]=$padre->[$i];
                                        }
                                                $intento++;
                        if($ContC[$hijo->[$i]] >= $Max )
                                {
                                        $color_invalido=1;
                                        print "Color $hijo->[$i] del padre invalido en c
```

```
                                        if DEBUG == 1;
                                                                }
                        else {
                                        for(my $j=0; $j<$i; $j++)
                                                {
                                                        if (defined $Mrp->[$j][$i] && $Mrp->[$j]
== 1 && $hijo->[$j] == $hijo->[$i])
                                                {
                                                        $color_invalido=1;
                                                        print "Color $hijo->[$i] del padre inval
                                                        if DEBUG == 1;
                                                }
                                                }
                                        }
                }

                        while ($color_invalido==1)
                                        {
                                        $color_invalido=0;
                                        $intento++;
                                        $hijo->[$i]=int(rand($obj->{Colores_Totales}))+1;

                                if($ContC[$hijo->[$i]] >= $Max ){

                                        $color_invalido=1;



INICIA_INDIVIDUO:


        for(my $i=0; $i<=$obj->{Colores_Totales}; $i++){
                $ContC[$i]=0;
        }
        my $Ind=[];

        for(my $i=0; $i<$N; $i++){


        my $color_invalido=0;

                my $intento=0;
                do{
                        $color_invalido=0;

                        $Ind->[$i]=int(rand($obj->{Colores_Totales}))+1;


                        if($ContC[$Ind->[$i]] >= $Max ){
                                $color_invalido=1;
                                print "Color $Ind->[$i] invalido en clase $i por exceder maximo de esos
```

```
                                        if DEBUG == 1;
                                } else {
                                        for(my $j=0; $j<$i; $j++){
                                        if (defined $Mrp->[$j][$i] && $Mrp->[$j][$i]  == 1 && $Ind->[$j]
                                                $color_invalido=1;
                                                print "Color $Ind->[$i] invalido en clase $i por ser igu
                                                        if DEBUG == 1;
                                        }
                                        }
                                }

                                $intento++;


                                $ContC[$Ind->[$i]]++ if $color_invalido == 0;



                                if ($intento >= 2*$obj->{Colores_Totales}){
                                        $individuo_eliminado++;
                                        if (DEBUG==1){                                          print "Se vuelve a crear
                                                print "\n\n-------Individuo Eliminado $individuo_eliminado:------
                                                print "$_ " foreach @$Ind;
                                                print "\n\n-------Finaliza Individuo Eliminado-------\n\n";
                                        }
                                        goto INICIA_INDIVIDUO;
                                }
                        } while ($color_invalido==1);
                }

sub Obtener_Restricciones {
                        my $obj=shift;
                        die "Matriz de Datos no Definida\n"
                                unless defined $obj->{M_Datos};

                my $Mrp=[];

                my $M=$obj->{M_Datos};
                my $N=@{$M};

                for(my $i=0; $i<$N; $i++){
                        for(my $j=$N-1; $j>$i; $j--){
                                if($M->[$i][0] eq $M->[$j][0] ){
                                        $Mrp->[$i][$j]=1;
                                } else {
                                        $Mrp->[$i][$j]=0;
                                }
                        }
                }

                if( DEBUG == 1){
                        print "\n\n-------Matriz de Restricciones y Penalizaciones (Solo Penalizaciones):------\n
```

```
                $obj->Imprimir_Matriz($Mrp);
                print "\n------Finaliza Restricciones y Penalizaciones (Solo Penalizaciones)------\n\n";
        }

        $obj->{M_Restricciones_Penalizaciones} = $Mrp;
        return $Mrp;
         }


sub Imprimir_Matriz {
        my $obj=shift;
        my $matriz=shift;

        for (my $i=0; $i<@{$matriz};$i++){

                print("\n"), next unless defined $matriz->[$i];
                for (my $j=0; $j<@{$matriz->[$i]}; $j++){
                        if (defined $matriz->[$i][$j]){
                                print "$matriz->[$i][$j] ";
                        } else {
                                 print "- "
                                }
                }
                print "\n";
        } }


sub Obtener_Matriz_de_Datos {
        my $obj=shift;
                my $M=[];

        my $n=0;
        print "\n\n------Matriz de Datos:------\n"
                if DEBUG == 1;
        while (my $l = <>){
                next if $l =~ /MATERIAS/;
                next if $l =~ /^\s+$/;
                chomp $l;

                my @m = split (/_/, $l);
                $m[1] =~ s/\{|\}//g;
                ($m[1], $m[2]) = split //, $m[1];
                $M->[$n]=\@m;
                print "$M->[$n][0] $M->[$n][1] $M->[$n][2]\n"
                        if DEBUG == 1;
                $n++;
        }
        print "\n------Finaliza Matriz de Datos------\n\n"
                if DEBUG == 1;

        $obj->{M_Datos}=$M;
        return $M;
```

```
}
```