



Universidad Nacional Mayor de San Marcos

Universidad del Perú. Decana de América

Facultad de Ingeniería de Sistemas e Informática

Escuela Académico Profesional de Ingeniería de Sistemas

Aplicación de algoritmos genéticos para el diseño de un sistema de apoyo a la generación de horarios de clases para la Facultad de Ingeniería de Sistemas e Informática de la UNMSM

TESINA

Para optar el Título Profesional de Ingeniero de Sistemas

AUTORES

Raúl Esteban NAUPARI QUIROZ

Gissela Katheryn ROSALES GERÓNIMO

ASESOR

Augusto CORTEZ VÁSQUEZ

Lima, Perú

2010



Reconocimiento - No Comercial - Compartir Igual - Sin restricciones adicionales

<https://creativecommons.org/licenses/by-nc-sa/4.0/>

Usted puede distribuir, remezclar, retocar, y crear a partir del documento original de modo no comercial, siempre y cuando se dé crédito al autor del documento y se licencien las nuevas creaciones bajo las mismas condiciones. No se permite aplicar términos legales o medidas tecnológicas que restrinjan legalmente a otros a hacer cualquier cosa que permita esta licencia.

Referencia bibliográfica

Naupari, R. & Rosales, G. (2010). *Aplicación de algoritmos genéticos para el diseño de un sistema de apoyo a la generación de horarios de clases para la Facultad de Ingeniería de Sistemas e Informática de la UNMSM*. Tesina para optar el título de Ingeniero de Sistemas. Escuela Académico Profesional de Ingeniería de Sistemas, Facultad de Ingeniería de Sistemas e Informática, Universidad Nacional Mayor de San Marcos, Lima, Perú.

Título:

APLICACIÓN DE ALGORITMOS GENÉTICOS PARA EL DISEÑO DE UN SISTEMA DE APOYO A LA GENERACIÓN DE HORARIOS DE CLASES PARA LA FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA DE LA UNMSM.

Bachilleres:

NAUPARI QUIROZ, Raúl Esteban

ROSALES GERÓNIMO, Gissela Katheryn

Tesina presentada a consideración del cuerpo docente de la Facultad de Ingeniería de Sistemas e Informática de la UNMSM como parte de los requisitos para obtener el Título Profesional de Ingeniero de Sistemas.

Aprobado por:

Nombre:

Presidente

Nombre:

Miembro del Jurado

Nombre:

Miembro Asesor

Lima – Perú
Febrero 2010

Resumen

Título: Aplicación de algoritmos genéticos para el diseño de un sistema de apoyo a la generación de horarios de clases para la Facultad de Ingeniería de Sistemas e Informática de la UNMSM.

Bachilleres: NAUPARI QUIROZ, Raúl Esteban
ROSALES GERÓNIMO, Gissela Katheryn

Asesor u Orientador de la Tesina: CORTEZ VASQUEZ, Augusto

Título o Grado académico a obtener: Ingeniero de Sistemas

El presente trabajo tiene como objetivo encontrar una solución al problema de elaboración de horarios de clases que existe en la Facultad de Ingeniería de Sistemas e Informática de la UNMSM. Este problema consiste en la asignación de un conjunto de recursos dentro de un número limitado de periodos de tiempo; además de satisfacer un conjunto de restricciones que se presentan en el ámbito académico. La importancia de resolver el problema radica en minimizar el tiempo y la cantidad de personal que interviene en el proceso de elaboración de horarios, además de brindar un apoyo en dicho proceso al personal responsable. Durante la investigación se realizó una comparativa entre algunas técnicas de búsqueda y optimización conocidas, tales como: la Búsqueda Tabú, los Algoritmos Meméticos, el Recocido Simulado y los Algoritmos Genéticos, para identificar cual de ellas posee las mejores características que permita resolver nuestra problemática. Una vez realizado esto, se determinó utilizar los Algoritmos Genéticos, los cuales pertenecen al grupo de técnicas metaheurísticas. Estas técnicas son métodos que consisten en desarrollar algoritmos eficientes que sean capaces de dar “buenas soluciones” a problemas, en donde encontrar el óptimo resulta costoso. En base a lo descrito, se propone un modelo matemático en el cual se definen las restricciones del problema, para luego establecer el diseño de la solución y la adaptación del algoritmo a ésta. Para la implementación del algoritmo se utilizó el lenguaje de programación Java; obteniéndose como resultado diferentes horarios de clases, los cuales son validados con los horarios elaborados manualmente; dicha comparación permitirá determinar el alcance y las limitaciones del algoritmo creado en este trabajo. Finalmente se llegó a la conclusión que el modelo desarrollado encontró buenas soluciones al problema dentro del margen de error definido.

Palabras claves: Algoritmos Genéticos, Timetabling, Optimización, Asignación de carga académica, Meta Heurísticas.

Abstract

Title: Application of genetic algorithms to design a support system to generate class schedules for the Faculty of Engineering and Computer Systems of San Marcos University.

Bachees: NAUPARI QUIROZ, Raúl Esteban
ROSALES GERÓNIMO, Gissela Katheryn

Advisor or Supervisor of Dissertation: CORTEZ VASQUEZ, Augusto

Qualification or degree to obtain: Systems Engineer

The present work aims to find a solution to the problem of development of class schedules that exists in the Faculty of Engineering and Computer Systems of San Marcos University. This problem can be defined as a set allocation of resources within a limited number of time periods, in addition to meeting a set of constraints that arise in academia. The importance of resolving the problem is to minimize the time and resources used in the scheduling process, in addition to providing support in this process to responsible staff. During the investigation it conducted a comparison between some search and optimization techniques known, such as Tabu Search, the memetic algorithms, simulated annealing and genetic algorithms to identify which of them has the best features that could solve our problems. Once this was determined using genetic algorithms, which belong to the group of metaheuristics techniques.

These techniques are methods that are to develop efficient algorithms that are capable of "good solutions" to problems, where finding the optimum is costly. These genetic algorithms are based on natural biological evolution, and can be adapted to solving many problems. Based on what has been described, we propose a mathematical model in which we define the constraints of the problem, then establish the design of the solution and adapting the algorithm to it. To implement the algorithm we used the Java programming language, obtaining as a result different class schedules, which are validated manually prepared schedules, such comparison will help determine the scope and limitations of the algorithm developed in this work. Finally, conclude that the model developed good solutions to the problem found within the defined margin of error.

Keywords: Genetic Algorithms, Timetabling, Optimization, Allocation of academic load, Meta Heuristics.

Ficha catalográfica

NAUPARI QUIROZ, Raúl Esteban

ROSALES GERÓNIMO, Gissela Katheryn

APLICACIÓN DE ALGORITMOS GENÉTICOS PARA EL DISEÑO DE UN
SISTEMA DE APOYO A LA GENERACIÓN DE HORARIOS DE CLASES PARA LA
FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA DE LA UNMSM

Inteligencia Artificial

(Lima 2010)

(UNMSM, Pregrado, Ingeniería de Sistemas e Informática)

Agradecimientos

A nuestros padres por su incondicional cariño, dedicación, comprensión y sobretodo por su paciencia, gracias por todos los consejos y el gran apoyo que nos han entregado no solamente en esta etapa de nuestra vida, sino también a lo largo de ésta. A ustedes dedicamos este trabajo.

A nuestros hermanos y demás seres queridos por estar siempre con nosotros, por su preocupación y gran cariño, gracias por estar siempre a nuestro lado.

A todos los que han hecho posible el desarrollo de este trabajo, gracias por mostrarnos siempre su apoyo incondicional.

Y sobre todo agradecemos a Dios por ser siempre un guía en nuestras vidas.

ÍNDICE

Resumen	iii
Abstract	iv
Ficha catalográfica	v
Agradecimientos	vi
Índice	vii
Índice de Tablas	xiii
Índice de Figuras	xiv
Introducción	1
1. Capítulo 1: Planteamiento Metodológico	3
1.1. Introducción	3
1.2. El Problema	5
1.2.1. Antecedentes del Problema	5
1.2.2. Realidad Problemática	7
1.2.3. Descripción del Problema	9
1.2.3.1. Enunciado del Problema	10
1.2.3.2. Restricciones para realizar horarios de clases	11
1.2.3.2.1. Restricciones Obligatorias	11
1.2.3.2.1.1. Restricciones Obligatorias Generales	11
1.2.3.2.1.2. Restricciones Obligatorias Específicas	12
1.2.3.2.2. Restricciones Deseables	13
1.2.3.2.2.1. Restricciones Deseables Generales	13
1.2.3.2.2.2. Restricciones Deseables Específicas	13
1.2.3.3. Definición Formal del Problema	13
1.2.3.3.1. Modelo General	14
1.2.3.3.1.1. Conjunto de Datos	14
1.2.3.3.1.2. Parámetros	20
1.2.3.3.1.3. Variables de decisión	20
1.2.3.3.1.4. Variable de duración	21
1.2.3.3.1.5. Modelamiento de Restricciones Obligatorias	21
1.2.3.3.1.6. Modelamiento de Restricciones Deseables	23
1.2.3.3.1.7. Función Objetivo	24
1.2.3.3.1.7.1. Función objetivo global	24
1.2.3.3.1.7.2. Función objetivo local	24
1.2.3.3.1.8. Criterio de Optimización	25
1.2.3.3.2. Modelo Particular	25

1.2.3.3.2.1. Conjunto de Datos	25
1.2.3.3.2.2. Parámetros	25
1.2.3.3.2.3. Variables de decisión	26
1.2.3.3.2.4. Variable de duración	26
1.2.3.3.2.5. Modelamiento de Restricciones Obligatorias	26
1.2.3.3.2.6. Modelamiento de Restricciones Deseables	27
1.2.3.3.2.7. Función Objetivo	28
1.2.3.3.2.7.1. Función objetivo global	28
1.2.3.3.2.7.2. Función objetivo local	28
1.2.3.3.2.8. Criterio de Optimización	28
1.2.3.4. Complejidad Computacional	28
1.2.3.4.1. Problemas P	29
1.2.3.4.2. Problemas NP	29
1.2.3.4.3. Problemas NP-Completo	30
1.3. Delimitaciones de la investigación	33
1.3.1. Delimitación Espacial	33
1.3.2. Delimitación Social	33
1.3.3. Delimitación Técnica	33
1.4. Justificación	33
1.4.1. Justificación de carácter práctico	33
1.4.2. Justificación de carácter metodológico	34
1.5. Objetivos	34
1.5.1. Objetivo General	34
1.6. Hipótesis General	34
1.7. Variables e Indicadores	34
2. Capítulo 2: Marco Teórico	36
2.1. El proceso de evolución	36
2.2. Introducción a los algoritmos genéticos	38
2.3. Definición de algoritmos genéticos	40
2.4. Analogía de un algoritmo genético con la naturaleza	42
2.5. Anatomía de un algoritmo genético	46
2.6. Componentes de un algoritmo genético	46
2.7. Características de los algoritmos genéticos	49
2.8. Algoritmo genético simple	50
2.8.1. Funcionamiento	51

2.8.2. Algoritmo básico	52
2.9. Codificación de variables	54
2.10. Parámetros de Configuración	55
2.10.1. Condición de Término	55
2.10.2. Tamaño de la Población	56
2.10.3. Porcentaje de Emparejamiento	56
2.10.4. Porcentaje de Mutación	56
2.11. Operadores Genéticos	57
2.11.1. Selección	57
2.11.1.1. Elitista	57
2.11.1.2. Proporcional a la aptitud	58
2.11.1.3. Escalada	58
2.11.1.4. Basado en rango	58
2.11.1.5. Rueda de ruleta	58
2.11.1.6. Por Torneo	59
2.11.1.7. Por Estado Estacionario	59
2.11.1.8. Generacional	59
2.11.1.9. Jerárquica	60
2.11.2. Emparejamiento	60
2.11.2.1. Emparejamiento de n-puntos	60
2.11.2.2. Emparejamiento de n-puntos	61
2.11.2.3. Emparejamiento segmentado	61
2.11.2.4. Emparejamientos Especializados	61
2.11.3. Mutación	61
2.11.3.1. Mutación de bit	62
2.11.3.2. Mutación multibit	62
2.11.3.3. Mutación de gen	62
2.11.3.4. Mutación multigen	62
2.11.3.5. Mutación de intercambio	63
2.11.3.6. Mutación de barajado	63
2.12. Ventajas y desventajas	64
2.13. Cómo saber si es posible usar un Algoritmo Genético	65
2.14. Ejemplo del funcionamiento de un algoritmo genético	66
2.14.1. Un Problema de ejemplo	66
2.14.2. Representación genética del ejemplo	66
2.14.3. Función de aptitud del ejemplo	67
2.14.4. Operadores del ejemplo	68

2.14.5. Resultados del ejemplo	68
3. Capítulo 3: Estado del Arte	70
3.1. Introducción	70
3.2. Métodos Existentes para Resolver el Problema	71
3.2.1. Búsqueda Tabú	71
3.2.1.1. Descripción	72
3.2.1.2. Aplicaciones de la búsqueda tabú al problema de la generación de horarios	79
3.2.2. Algoritmos Meméticos	86
3.2.2.1. Métodos de Optimización Local	89
3.2.2.2. Anatomía de un Algoritmo Meméticos	90
3.2.2.3. Aplicaciones de los Algoritmos Meméticos al problema de la generación de horarios	96
3.2.3. Recocido simulado	99
3.2.3.1. Estructura del algoritmo	101
3.2.3.2. Aplicaciones del Recocido Simulado al problema de la generación de horarios	106
4. Capítulo 4: Aporte Teórico	109
4.1. Introducción	109
4.2. Evaluación y Selección del Algoritmo	110
4.2.1. Cuadro comparativo de Metaheurísticas	110
4.2.2. Análisis de tiempo y calidad de solución	113
4.2.3. Comparación de tiempos	115
4.3. Adaptación del algoritmo	117
4.3.1. Unidad de asignación	117
4.3.2. Separación en Sub-Fases	119
4.3.3. Representación del problema	122
4.3.4. Codificación mediante alfabetos duros	123
4.3.5. Inicialización de la población	125
4.3.6. Función de evaluación y tratamientos de las restricciones	128
4.3.7. El proceso de selección	130
4.3.7.1. Escalamiento de la aptitud	131
4.3.7.2. Selección de torneo	132
4.3.8. Criterio de Término	133
4.3.9. Operadores genéticos	134

4.3.9.1. Operador de emparejamiento	134
4.3.9.2. Operador de mutación	135
4.3.10. Reinserción y Elitismo	136
5. Capítulo 5: Diseño de la Solución.	137
5.1. Modelado del Negocio	137
5.1.1. Diagramas de Actividades	140
5.2. Vista de Casos de Uso	144
5.2.1. Diagrama de Casos de Uso	144
5.2.2. Especificaciones de Casos de Uso y Diagramas de Secuencia	145
5.2.2.1. Caso de Uso “Ingresar al Sistema”	145
5.2.2.2. Caso de Uso “Cargar Información de Horas Académicas”	148
5.2.2.3. Caso de Uso “Cargar Información de Cursos”	151
5.2.2.4. Caso de Uso “Cargar Información de Docentes”	156
5.2.2.5. Caso de Uso “Cargar Información de Aulas”	161
5.2.2.6. Caso de Uso “Configurar Valores de Parámetros”	166
5.2.2.7. Caso de Uso “Generar Horario Académico”	169
5.3. Diagrama de Clases	171
5.4. Modelo de Datos	174
5.5. Arquitectura de la Solución	175
5.5.1. Restricciones no funcionales	176
5.6. Vista de Componentes	177
5.7. Vista de Despliegue	178
5.8. Prototipos de la Solución	179
5.8.1. Ventana de Ingreso al Sistema	179
5.8.2. Ventana de Carga de Información de Horarios	181
5.8.3. Ventana de Carga de Información de Cursos	181
5.8.4. Ventana de Carga de Información de Docentes	182
5.8.5. Ventana de Carga de Información de Aulas	183
5.8.6. Ventana de Configuración de Valores de Parámetros	184
5.8.7. Ventana de Generación de Horario Académico	185
6. Capítulo 6: Pruebas y Resultados	186
6.1. Introducción	186
6.2. Casos de Análisis	187
6.3. Esquema de Pruebas	188
6.3.1. Pruebas sobre el proceso de selección	189

6.3.1.1. Prueba de escalamiento de la aptitud	189
6.3.1.2. Prueba sobre el método de selección	192
6.3.1.3. Prueba sobre el tamaño del torneo	192
6.3.2. Pruebas sobre los operadores genéticos	193
6.3.2.1. Prueba sobre el método de emparejamiento	194
6.3.2.1.1. Prueba de porcentajes de emparejamiento	196
6.3.2.2. Prueba sobre el método de mutación	197
6.3.2.2.1. Prueba de porcentaje de mutación	197
6.3.3. Prueba sobre el proceso de reinserción	199
6.3.3.1. Prueba sobre el método de reinserción	199
6.3.3.2. Prueba sobre el uso del elitismo	200
6.3.4. Prueba de parámetros de configuración	201
6.3.4.1. Prueba de número de individuos	202
6.3.4.2. Prueba de número de generaciones	203
6.4. Horario de Clases	204
7. Capítulo 7: Conclusiones y Recomendaciones	217
7.1. Conclusiones	217
7.2. Recomendaciones	219
Bibliografía	221
Apéndice 1: Definición de términos usados	223
Apéndice 2: Definición de acrónimos y abreviatura	226
Apéndice 3: Estructura de Datos del Modelo General	227
Apéndice 4: Descripción de paquetes del API JGAP	228

Índice de Tablas

Tabla 1.1. Tiempos de los algoritmos	31
Tabla 2.1. Datos obtenidos con la investigación de mercado en cada una de las regiones	67
Tabla 4.1. Puntuaciones para calificar las características de las metaheurísticas	112
Tabla 4.2. Resultados de la calificación de las metaheurísticas	112
Tabla 4.3. Medición de soluciones en intervalos de tiempo	114
Tabla 4.4. Habilidad de las meta heurísticas de encontrar la mejor solución	115
Tabla 4.5. Habilidad de las metaheurísticas de producir un máximo costo de reducción	115
Tabla 4.6. Comparación de tiempos para alcanzar una adecuada solución	116
Tabla 4.7. Horas de Dictado de Algorítmica I	118
Tabla 4.8. Clases generadas para el curso de Algorítmica I	119
Tabla 6.1. Valores para realizar el escalamiento de la aptitud	189
Tabla 6.2. Porcentaje de ocurrencia del operador de emparejamiento	195
Tabla 6.3. Porcentaje de ocurrencia del operador de mutación	197

Índice de Figuras

Figura 2.1. Individuo genético binario	51
Figura 2.2. Funcionamiento de un algoritmo genético	52
Figura 2.3. Pseudocódigo de un algoritmo genético	53
Figura 2.4. Construcción de Ruleta con probabilidades	59
Figura 2.5. Esquemmatización de Selección por Torneo	59
Figura 2.6. Emparejamiento de 1 punto	61
Figura 2.7. Mutación de un bit	62
Figura 2.8. Cadena representativa de un cromosoma del ejemplo	68
Figura 2.9. Gráfica de los valores de la Tabla	69
Figura 3.1. Vecindario de la solución actual	73
Figura 3.2. Trayectoria para encontrar la solución óptima	74
Figura 3.3 Lista Tabú	74
Figura 3.4 El componente de memoria de corto plazo de la búsqueda tabú	76
Figura 4.5. Selección del mejor movimiento posible	77
Figura 3.6. Algoritmo de la Búsqueda Tabú Simple	79
Figura 3.7. Estructura de un horario de clases	80
Figura 3.8. Ilustración del movimiento(a)	81
Figura 3.9. Ilustración del movimiento (b)	81
Figura 3.10. Estructura de un Algoritmo Memético	88
Figura 3.11. Esquema de una generación	91
Figura 3.12. Generación de la nueva población	91
Figura 3.13. Esquema de un optimizador local	93
Figura 3.14. El esqueleto genérico de un MA	94
Figura 3.15. Generación guiada de la población inicial	95
Figura 3.16. Reiniciación de la población en un Algoritmo Memético	95
Figura 3.17. Representación de la solución	97
Figura 3.18. Diagrama de Flujo del Algoritmo de Recocido Simulado	101
Figura 4.1. Comportamiento del tiempo de operación de las metaheurísticas	113
Figura 4.2. Estructura de la unidad de asignación (clase)	117
Figura 4.3. Fases para la asignación de clases a los recursos	120
Figura 4.4. Representación vectorial del cromosoma	122
Figura 5.1. Diagrama de Casos de Uso del Negocio	140
Figura 5.2. Diagrama de Actividad de Recopilar Información de Docentes	141
Figura 5.3. Diagrama de Actividad de Establecer grupos aperturados por curso	141
Figura 5.4. Diagrama de Actividad de Establecer Disponibilidad de Aulas	142

Figura 5.5. Diagrama de Actividad de Elaborar Horario Académico	142
Figura 5.6. Diagrama de Actividad de Verificar Carga Académica	143
Figura 5.7. Diagrama de Actividad de Publicar Horarios Académicos	143
Figura 5.8. Diagrama de Casos de Uso del Sistema	144
Figura 5.9. Diagrama de Secuencia de “Ingresar al Sistema”	147
Figura 5.10. Diagrama de Secuencia de “Cargar Información de Horas Académicas”	150
Figura 5.11. Diagrama de Secuencia de “Cargar Información de Cursos”	155
Figura 5.12. Diagrama de Secuencia de “Cargar Información de Docentes”	160
Figura 5.13. Diagrama de Secuencia de “Cargar Información de Aulas”	165
Figura 5.14. Diagrama de Secuencia de “Configurar Valores de Parámetros”	168
Figura 5.15. Diagrama de Secuencia de “Generar Horario Académico”	171
Figura 5.16. Diagrama de clases y sus relaciones	172
Figura 5.17. Diagrama Entidad – Relación	175
Figura 5.18. Arquitectura del Sistema	176
Figura 5.19. Diagrama de Componentes	177
Figura 5.20. Estructura de paquetes de JGAP 3.4.4	178
Figura 5.21. Diagrama de Despliegue	179
Figura 5.22. Prototipo del Caso de Uso “Ingresar al Sistema”	180
Figura 5.23. Prototipo del Menú Principal	180
Figura 5.24. Prototipo del Caso de Uso “Cargar Información de Horarios”	181
Figura 5.25. Prototipo del Caso de Uso “Cargar Información de Cursos”	182
Figura 5.26. Prototipo del Caso de Uso “Cargar Información de Docentes”	183
Figura 5.27. Prototipo del Caso de Uso “Cargar Información de Aulas”	184
Figura 5.28. Prototipo del Caso de Uso “Configurar Valores de Parámetros”	185
Figura 5.29. Prototipo del Caso de Uso “Generar Horario Académico”	185
Figura 6.1. Número de generaciones vs. Escalamiento de la aptitud utilizado	190
Figura 6.2. Frecuencia de éxito vs. Escalamiento de la aptitud utilizado	190
Figura 6.3. Número de generaciones vs. Método de selección utilizado	191
Figura 6.4. Número de generaciones vs. Tamaño de torneo utilizado	192
Figura 6.5. Frecuencia de éxito vs. Tamaño de torneo utilizado	192
Figura 6.6. Número de generaciones vs. Método de emparejamiento utilizado	194
Figura 6.7. Frecuencia de éxito vs. Método de emparejamiento utilizado	194
Figura 6.8. Número de generaciones vs. Porcentaje de emparejamiento	196
Figura 6.9. Número de generaciones vs. Porcentaje de mutación	197
Figura 6.10. Frecuencia de éxito vs. Porcentaje de mutación	198
Figura 6.11. Número de generaciones vs. Método de reinserción utilizado	199

Figura 6.12. Frecuencia de éxito vs. Método de reinserción utilizado	199
Figura 6.13. Número de generaciones vs. Elitismo	200
Figura 6.14. Frecuencia de éxito vs. Tamaño de la población inicial - Fase uno	201
Figura 6.15. Frecuencia de éxito vs. Tamaño de la población inicial - Fase dos	202
Figura 6.16. Frecuencia de éxito vs. Tamaño de la población inicial - Fase tres	202
Figura 6.17. Número de generaciones vs. Fase	203

Introducción

La asignación de recursos limitados, sujetos a restricciones de distinto tipo, ha sido un problema de relevancia en áreas como la investigación operativa, la administración de las organizaciones e incluso la inteligencia artificial. Por lo general, la solución de este tipo de problemas trae consigo una serie de condiciones de eficiencia, tiempo y oportunidad que deben ser tomadas en cuenta, más allá de la forma correcta de realizar la asignación. Es deseable considerar todas las variables al mismo tiempo en un problema de asignación, ponderándolas según su importancia, y buscando obtener el mayor beneficio en la resolución del problema. De esta manera, y como revisar un espacio de combinaciones muy grande es difícil, conviene dirigir o conducir la búsqueda de la solución que permita resolver el problema de forma eficiente.

Podemos considerar varios métodos de resolución como puede ser la búsqueda nodo por nodo de forma sistemática [10], que en algún momento debieran encontrar la solución, pero nada puede garantizar que no se deba recorrer todo el espacio de estados para lograrlo. Si bien se cuenta con métodos de ramificación y acotación (que se utilizan en problemas de programación lineal entera), los valores en las restricciones varían con el tiempo, haciendo que se deba calcular las configuraciones constantemente.

Otra alternativa es utilizar métodos de búsqueda heurística [10], donde se sacrifica la completitud de la exploración por la eficiencia en el tiempo de respuesta. Estos métodos necesitan contar con información de qué tan lejos estamos de una posible solución para poder conducir la búsqueda y, por lo general, la determinación de una buena función heurística que contemple varias variables no es simple de conseguir.

La propuesta que se hace en este trabajo es utilizar algoritmos genéticos para la resolución del problema de generación de horarios de clases, tomando como dominio del problema los cursos que se dictan en la Facultad de Ingeniería de Sistemas e Informática de la Universidad Nacional Mayor de San Marcos.

Durante los últimos años se ha experimentado un creciente interés por la construcción de sistemas de resolución de problemas basados en el proceso de evolución natural. En este sentido los algoritmos genéticos son algoritmos de búsqueda y optimización cuyo mecanismo de funcionamiento está basado en dicho proceso.

Los objetivos son maximizar el uso de los recursos, minimizar el desperdicio del espacio físico y establecer adecuadamente la carga académica de los docentes para lograr la asignación completa de los horarios de clases.

El presente trabajo está organizado en 7 capítulos, en el capítulo uno se presenta la descripción del problema a través de un modelo matemático, su función objetivo, las restricciones, las variables y el modelo particular que es aplicado al problema práctico, además se describe las delimitaciones de la investigación, la justificación, los objetivos, la hipótesis, y algunos conceptos, tales como, complejidad computacional. En el capítulo dos se describen los conceptos teóricos relacionados a algoritmos genéticos, incluyendo sus componentes, características, el funcionamiento, los operadores genéticos y su aplicación en un problema de ejemplo. En el capítulo tres se detallan otros métodos empleados hasta la actualidad para resolver el problema planteado, así como un estudio comparativo entre dichos métodos. En el capítulo cuatro se presenta el aporte teórico de la investigación, mientras que en el capítulo cinco se especifica el diseño de la solución al problema. En el capítulo seis se muestran las pruebas realizadas a cada una de las etapas del algoritmo genético, tales como, pruebas sobre el proceso de selección, sobre operadores genéticos y sobre el proceso de reinserción; además se muestran los resultados obtenidos por el sistema desarrollado. Por último en el capítulo siete se detallan las conclusiones surgidas durante el desarrollo de la investigación y las recomendaciones para posibles trabajos futuros.

Capítulo 1: Planteamiento Metodológico

En este capítulo se describirá el planteamiento del problema, las limitaciones y la justificación de la investigación, así como los objetivos planteados y se realizará el planteamiento de hipótesis y se identificarán las variables y los indicadores de cada una de ellas.

1.1 Introducción

A través de los años, la sociedad ha presentado avances en diferentes aspectos que la han tornado más compleja, en parte debido al nivel de aprendizaje de los individuos en las universidades. En respuesta a esto, las universidades han ido creciendo en una complejidad enigmática, en diversidad de operaciones y en la capacidad de responder a las necesidades educativas de hoy. El objetivo de los estudiantes de aprender más, de una amplia gama de profesionales que pueden contribuir a su conocimiento, ha hecho que sean cada vez más exigentes al momento de elegir una universidad. Con lo cual las universidades, como cualquier negocio, se han visto en la necesidad de re-diseñar sus procesos internos para hacerlos más eficientes y así satisfacer al exigente alumnado de hoy.

Actualmente las instituciones académicas presentan diversas dificultades para resolver el problema de asignación horaria, debido a la cantidad de restricciones que se presentan y el criterio con el que se aplican. La complejidad de este proceso radica en las restricciones asociadas a los recursos, tales como: docentes, alumnos, cursos, aulas, etc.

Este problema es conocido con el nombre de Timetabling, que consiste en la asignación de entidades o recursos escasos dentro de un número limitado de períodos de tiempo; y es utilizada en distintas áreas para regular diferentes operaciones donde sea necesaria la ubicación de un recurso en un sitio determinado en una hora específica, como por ejemplo en el transporte (vuelos aéreos), en salud (programación de cirugías), en el campo del entretenimiento (deportes), etc.

En general, han habido muchas tentativas de tratar este problema usando métodos clásicos, como por ejemplo la programación entera, pero sin mucho éxito. También han habido otras un poco más acertadas como las de tratar el problema usando varios métodos de búsqueda heurística.

Hasta la fecha, en la Escuela de Ingeniería de Sistemas e Informática de la Facultad de Ingeniería de Sistemas e Informática (FISI) de la Universidad Nacional Mayor de San Marcos (UNMSM), el proceso de elaboración de horarios de clases todavía se hace manualmente, porque se deben considerar restricciones que son difíciles de modelar y automatizar. En este contexto, estudiamos los Algoritmos Genéticos (AG) para encontrar la solución a dicho problema. Esta técnica ha demostrado resultados prometedores en muchos campos de la optimización en donde se modelaban y diseñaban métodos, y donde el tamaño y la dimensión son factores complicados.

Este trabajo tiene como objetivo alcanzar una gran utilidad, a través de dos maneras concretas, para solucionar el problema de generación de horarios de clases de la facultad de tal manera que se pueda encontrar una óptima solución: En primer lugar, incorporar tantas restricciones posibles y realistas del problema, para encontrar una óptima solución a través de una adecuada asignación de recursos. Y en segundo lugar, proporcionar el diseño de una aplicación fácil de utilizar para el personal de la FISI.

Específicamente este trabajo desea diseñar un sistema práctico y útil para generar los horarios de clases, el cual resulta ser un trabajo demasiado arduo y complejo. Por supuesto, el estudio está dirigido a nuestra facultad, pero este método puede fácilmente extenderse a otras facultades e incluso universidades, dado que se encuentran en condiciones similares en lo que respecta al proceso de generación de horarios.

1.2 El Problema

En esta sección se detallarán los antecedentes del problema, la realidad problemática, la descripción del problema, las restricciones para realizar los horarios de clase, el modelo general y particular del problema y la complejidad computacional.

1.2.1. Antecedentes del Problema

En toda institución de educación superior, al inicio de cada período académico se presenta la necesidad de asignar y coordinar los recursos económicos, materiales y humanos en beneficio de los estudiantes. Considerándose de vital importancia la organización y distribución de los horarios de clases de docentes y alumnos, para lo cual se debe tomar en cuenta las condiciones particulares originadas por el avance académico de los alumnos y algunos otros factores tales como la disponibilidad de docentes y la capacidad de aulas; que son las que complican aún más la programación de los horarios de clases.

El proceso de distribución horaria (Timetabling) es parte de los problemas de programación en general, dicha programación trata de la asignación satisfactoria de una cantidad de recursos para alcanzar los objetivos de una organización. Además es un procedimiento de toma de decisión, con la intención de optimizar objetivos.

Este proceso difiere de acuerdo al tipo de institución involucrada, ya sea una universidad o una escuela [26]:

- Asignación de horarios escolares, también conocida como Class-Teacher Problem, considera el horario semanal para las sesiones de los cursos de una escuela o colegio. Dado los cursos, docentes y una matriz

de requerimientos (que establece el número de sesiones que cada docente dicta por curso), el problema consiste en asignar las sesiones a los períodos de tiempo, de tal manera que ningún docente o curso tenga más de una sesión en el mismo período y que todas las sesiones del curso estén presentes en el horario.

- Asignación de horarios universitarios, consiste en organizar un horario para las clases de un conjunto de cursos, considerando un número determinado de aulas y bloques de tiempo.

La principal diferencia entre un horario escolar y uno universitario es la forma como se considera a los estudiantes. En el ámbito escolar éstos pueden considerarse como una entidad, debido a que es un grupo de alumnos que toman los mismos cursos. En el caso universitario, generalmente con régimen semiflexible, los estudiantes toman distintos cursos, por lo que se generan cursos en común con otros estudiantes.

Otra diferencia que se presenta son los docentes. En las escuelas se encargan de enseñar un curso y en la universidad generalmente imparten de 1 a 3 cursos.

Además, se presenta el problema de la capacidad de las aulas ya que cada curso tiene asociado su propio requerimiento; por el contrario, para el caso de las escuelas se pueden destinar todas las aulas como aptas para cualquier curso.

Por otro lado, existen diversas técnicas para la resolución de problemas de generación de horarios, los cuales se clasifican en dos grupos:

- Las técnicas tradicionales; son métodos que debido a su forma de buscar soluciones se les denomina completos, ya que recorren todo el espacio de búsqueda, es decir, encuentran todas las soluciones posibles a un determinado problema, sin embargo, estos dependen del número de variables que influyen en el problema, es decir, el éxito de estas depende directamente del número de variables que intervienen en el problema. Entre estas técnicas tenemos los siguientes tipos: Programación entera, Programación Lineal, Tableau, entre otras.

- Las técnicas no tradicionales; utilizadas principalmente en el campo de la inteligencia artificial, siendo una de las más conocidas las metaheurísticas, que consisten en sistematizar ideas con el fin de desarrollar algoritmos eficientes que sean capaces de entregar buenas soluciones a problemas, donde encontrar el óptimo resulta muy costoso e incluso en algunos casos imposible. Estos problemas se clasifican en diversas complejidades: En primer lugar tenemos la complejidad P, que son algoritmos que encuentran siempre una solución a un problema en tiempo polinomial y las de complejidad NP, que encuentran solución a un problema en tiempo polinomial usando una computadora no determinística.

En el ámbito de las ciencias computacionales, los problemas de tipo timetabling están clasificados como NP completo [10]-[3]. Un gran número de problemas tratables tienen un estado hasta ahora indeterminado; se supone que son intratables, pero esto no se ha demostrado. Entre la clase de problemas NP encontramos el problema de satisfactibilidades (SAT). Este es un problema central en la lógica matemática y la teoría de la computación. SAT es fundamental para la solución de varios problemas en razonamiento automático, diseño y manufactura asistida por computadoras, visión computacional, bases de datos, robótica, diseño de circuitos integrados, arquitectura de computadoras, redes de computadoras y asignación de tareas, entre otros. Los métodos para resolver el problema SAT juegan un papel crucial en el desarrollo de sistemas informáticos eficaces y las aplicaciones de dicho problema [16]-[2]. El caso de programación de horarios académicos es un ejemplo de este tipo de problemas. Las técnicas metaheurísticas combinan la simplicidad de sus ideas con su gran eficiencia para obtener buenos resultados para este tipo de problemas complejos. Entre éstas se encuentran: Simulated annealing (Simulado recocido), Genetic algorithms (Algoritmos genéticos), Tabú search (Búsqueda tabú), GRASP (Algoritmos Voraces), Neuronal networks (Redes neuronales), entre otras.

1.2.2. Realidad Problemática

Actualmente, como caso particular, en la Facultad de Ingeniería de Sistemas e Informática la tarea de elaboración de horarios académicos se realiza de

forma manual, estimando y estipulando los recursos como mejor parecen ajustarse según el criterio del responsable de elaborar dichos horarios en cada semestre académico. Esta tarea normalmente requiere varios días de trabajo; sin embargo, debido al volumen de datos y variables que intervienen en el proceso, la tarea resulta ser costosa, compleja y no siempre culmina en resultados satisfactorios; como por ejemplo, realizar una asignación de aulas que permanecen medianamente vacías ya que la cantidad de alumnos matriculados es menor a su capacidad o viceversa, es decir, tal vez se asignan recintos muy pequeños para una gran cantidad de estudiantes, etc. Además se puede producir insatisfacción en algunos aspectos, tales como, que un estudiante no podrá matricularse en todos los cursos que desea debido a que están programados en el mismo periodo de tiempo, o que a un docente se le programan dos cursos en el mismo horario o periodo, etc. Debido a esto se tiene como resultado varios problemas de funcionamiento, molestias personales, desprogramación, aunque lo más preocupante es la cantidad de tiempo que se emplea para solucionar los cruces de horarios que se puedan originar.

Para realizar la asignación de horarios, los encargados no cuentan con una planeación exacta de la información a usar, como por ejemplo, el número de estudiantes por curso, o la cantidad de grupos que se debe aperturar por cada curso; lo cual trae como consecuencia que al tener una mayor demanda de alumnado en un determinado curso se tenga que abrir un grupo adicional, asignarle un docente disponible y programarlos en un nuevo horario que beneficie a todos.

Con una asignación de horarios de clases automatizada se elimina el proceso manual, la cual requiere mucho tiempo y una desgastante y compleja coordinación entre directores de oficinas, coordinadores, etc., además se logra una adecuada administración de los espacios físicos de la facultad para evitar que se desaprovechen dichos espacios.

El proceso de elaboración de horarios de clases que a simple vista parece la optimización de sólo algunos parámetros como cantidad de alumnos por clase, capacidad de las aulas, cantidad de aulas disponibles, etc., ha sido motivo de varios trabajos de investigación. Para esto será necesario implementar una solución que tome en cuenta las limitaciones dinámicas y

múltiples, considerando además el conflicto de objetivos existentes que permita un modelo más cercano a la realidad.

Por último, cabe mencionar, que el modelado de las relaciones complejas entre los distintos parámetros del problema como son las aulas, los docentes con sus periodos disponibles, los cursos que pueden dictar, los periodos hábiles del ciclo académico y otros más, plantean un tipo de problema no resuelto hasta la actualidad de manera conjunta e integral, en parte por la escasez de investigación en el área.

1.2.3. Descripción del Problema

La elaboración de los horarios de clases tiene como finalidad brindar un conjunto de cursos que puede y debe cursar el alumno dependiendo del semestre académico a inscribirse y de las circunstancias académicas en que se encuentre. La manera en que se presentan dichos horarios y la libertad para escogerlos, será de gran ayuda en la realización y consolidación de su carrera, siempre y cuando estos horarios estén bien elaborados. Por el contrario, si no lo están puede llegar a repercutir negativamente en el alumnado de diferentes maneras, por ejemplo, un alumno puede llegar a estar rezagado en uno o más semestres si es que los grupos aperturados no poseen cupos disponibles. Otro problema se presenta cuando un alumno que selecciona un curso en un horario o periodo específico y con un determinado docente, se encuentra posteriormente con que el horario cambió o que el dictado del curso estará a cargo de otro docente o ambas cosas.

El problema de elaboración de horarios de clases se asocia a la ardua labor de estructurar los horarios de clases a partir de un conjunto de recursos, considerando cada una de las restricciones que tiene el problema y buscando maximizar el número total de períodos asignados en espacios de tiempo deseados. Es necesario considerar algunos factores como la carga académica de los docentes, las aulas disponibles para la impartición de clases y algunas otras políticas particulares que son definidas por la institución.

En cualquier problema de optimización existen objetivos, decisiones que tomar, recursos disponibles y necesidades. En el problema de programación de horarios universitarios, los recursos disponibles son los docentes, los estudiantes, los cursos que son parte del plan académico, los periodos hábiles semanales y las aulas de clase. Una solución para este problema debe agrupar estos recursos para producir un horario de clases, que cumpla con el conjunto de restricciones que serán fijadas por el personal responsable.

Es importante señalar que en muchas aplicaciones prácticas algunas de estas restricciones tienen carácter de necesarias; es decir, todas las soluciones factibles y significativas deben satisfacer este grupo de restricciones, tales como, la restricción que dos clases distintas no se pueden llevar a cabo en el mismo periodo de tiempo y en la misma aula de clase. También hay otras restricciones que se podrían denominar como altamente deseables, que podrían afectar en cierto grado la satisfacción total del horario que se elaborará. Una restricción altamente deseable sería, por ejemplo, no tener ninguna clase durante la hora del almuerzo; sin embargo, otras restricciones más importantes podrían forzarnos a programar clases sobre las horas de almuerzo, y a pesar de ello podría considerarse tolerable. Por lo tanto, el objetivo de un horario de clases sería satisfacer en gran medida varias restricciones que se encuentran en conflicto o involucradas entre sí.

1.2.3.1. Enunciado del Problema

Por consiguiente, observamos que el problema general encontrado en la FISI con relación a la generación de horarios de clases es: **¿En qué medida se puede automatizar el proceso de generación de horarios de clase, mediante el uso de algoritmos genéticos, con el fin de reducir el tiempo y los recursos empleados en su elaboración y a su vez elevar el grado de satisfacción del alumnado y el docente?**

1.2.3.2. Restricciones para realizar horarios de clases

Larrosa [7] define el término restricciones como condiciones que debe satisfacer el horario generado; y las clasifica en dos tipos:

- ✓ **Restricciones obligatorias**, son propiedades espaciales o temporales. Toda restricción obligatoria debe cumplirse, la violación de alguna origina un horario no válido. Podemos entenderlas también como reglas que deben satisfacerse.
- ✓ **Restricciones deseables** son restricciones que en realidad denotan preferencias del usuario (políticas flexibles) y se desea que se cumplan en la medida de lo posible. La violación de algunas de ellas seguirá produciendo un horario válido, pero de menor calidad que si se cumplieran todas éstas.

Adicionalmente clasificaremos cada uno de estos tipos en **Restricciones generales** las cuales podrán aplicarse a la mayoría de problemas de timetabling universitario y las **Restricciones específicas** que son propias de una sola institución, en este caso la FISI.

Para la solución de nuestro problema sólo se considerarán las restricciones obligatorias.

1.2.3.2.1. Restricciones Obligatorias

1.2.3.2.1.1. Restricciones Obligatorias Generales

- ✓ ROG1. Un curso debe tener asignado a lo más un docente en un aula en un periodo específico.
- ✓ ROG2. Un docente debe tener asignado a lo más un curso en un aula en un periodo específico.
- ✓ ROG3. Un aula puede tener a lo más un curso asignado y un docente en un periodo específico.
- ✓ ROG4. No es posible asignar a un docente un curso por el cual no posea preferencia.
- ✓ ROG5. No se puede asignar un curso a un docente fuera de su disponibilidad de tiempo.

- ✓ ROG6. La cantidad de estudiantes en un curso no puede sobrepasar la capacidad del aula.
- ✓ ROG7. Un curso debe cumplir con una cantidad de horas semanales requeridas. Según sea el caso, si el número de horas académicas es par se asignará dos horas diarias hasta cubrir la totalidad, sino se asignará un día de tres horas y el resto de dos horas. A cada uno de estos grupos de horas se les llamará bloques.
- ✓ ROG8. Las horas de un bloque para un determinado curso en un mismo día deben de ser consecutivas además de ser asignados a un docente y aula.
- ✓ ROG9. Los cursos no deben de solaparse entre ellos, ni en duración ni ubicación física, una vez seleccionada la hora de inicio de éstos y el aula correspondiente.

1.2.3.2.1.2. Restricciones Obligatorias Específicas

- ✓ ROE1. Un docente debe de impartir una cantidad determinada de horas de dictado de clases como máximo y como mínimo según su categoría.
- ✓ ROE2. Los cursos de teoría y práctica deben de asignarse a aulas de tipo no laboratorio.
- ✓ ROE3. Los cursos de laboratorio deben de asignarse a aulas de tipo laboratorio.
- ✓ ROE4. No deben coincidir los horarios de los cursos que corresponden a un mismo semestre y grupo.
- ✓ ROE5. Pueden existir asignaciones previas de cursos a un determinado periodo y aula.
- ✓ ROE6. Pueden existir asignaciones previas de docentes a un determinado período y curso.
- ✓ ROE7. Los cursos son clasificados en los turnos de mañana, tarde y noche según el semestre al que pertenecen.

1.2.3.2. Restricciones Deseables

1.2.3.2.1. Restricciones Deseables Generales

- ✓ RDG1. Dos cursos de semestres consecutivos no deben de ser asignados en un mismo periodo de tiempo.
- ✓ RDG2. Los bloques de un curso deben de distribuirse lo mejor posible en la semana.
- ✓ RDG3. Se debe minimizar el número de horas libres entre el fin de una clase y el inicio de otra.
- ✓ RDG4. Pueden existir prioridades o penalidades sobre cursos para que sean dictados en periodos determinados. Es decir, una solución será de mayor calidad mientras se cumplan adecuadamente las prioridades o se tenga la menor cantidad de asignaciones a períodos penalizados.
- ✓ RDG5. Cada bloque de un determinado curso sólo puede asignarse a lo más una sola vez cada día.

1.2.3.2.2. Restricciones Deseables Específicas

- ✓ RDE1. No se deben de asignar cursos en horario de almuerzo

1.2.3.3. Definición Formal del Problema

Una vez planteado el problema de timetabling para la FISI, vamos a presentar el modelo matemático general y específico del problema, en el cual presentamos los conjuntos de datos, los parámetros, las variables de decisión, las restricciones, la función objetivo y el criterio de optimización, que son necesarios para encontrar una solución factible, y en el mejor de los casos, una solución óptima.

Para formular un modelo matemático, que sirva de base para la solución del problema de timetabling por lo menos en un sentido aproximado, debemos considerar todas las intersecciones posibles existentes entre los cuatro elementos básicos que lo componen: aulas, docentes, cursos y periodos de tiempo.

1.2.3.3.1. Modelo General

En el apéndice 3 se muestra la visión global de la estructura de datos que se describe a continuación:

1.2.3.3.1.1 Conjunto de Datos

- ✓ Sea $C = \{1, \dots, c\} \forall c \in \mathbb{Z}$ el vector de cursos académicos.

Ejemplo: $C = \{$

Algorítmica I	Cálculo I	...	Base de Datos	...
---------------	-----------	-----	---------------	-----

$\}$

- ✓ Sea $G = \{g_1, \dots, g_n\}$ el vector de grupos aperturados para los cursos. Cada $g \in G$ contiene un conjunto de cursos $C_g \subseteq C$, que no pueden ser programados en un mismo periodo de tiempo. Dos grupos distintos pueden tener cursos iguales, por ejemplo sea C_{g_1} y C_{g_2} dos cursos pertenecientes a los grupos G_1 y G_2 , con $G_1 \neq G_2$ puede ocurrir que $C_{g_1} \cap C_{g_2} \neq \emptyset$.

Ejemplo: $G = \{$

Grupo 1	Grupo 2	Grupo 3
---------	---------	---------

$\}$

- ✓ Sea $L = \{l_1, \dots, l_n\}$ el vector de tipos que indica la modalidad del dictado de los cursos. Cada $l \in L$ contiene un conjunto de cursos $C_l \subseteq C$, que no pueden ser programadas en un mismo periodo de tiempo. Dos tipos distintos pueden tener cursos iguales, por ejemplo sea C_{l_1} y C_{l_2} dos cursos pertenecientes a los tipos L_1 y L_2 , con $L_1 \neq L_2$ puede ocurrir que $C_{l_1} \cap C_{l_2} \neq \emptyset$. Definimos de antemano $L = \{l_t, l_p, l_l\}$, cada elemento correspondiente a los cursos de tipo de teoría, práctica y laboratorio.

Ejemplo: $L = \{$

Teoría	Práctica	Laboratorio
--------	----------	-------------

$\}$

- ✓ Sea $T = \{1, \dots, t\} \forall w \in Z$ el vector de periodos de tiempo en el que se puede dictar un curso cualquiera $c \in C$.

Ejemplo: $T = \{$

Periodo 1	Periodo 2	..	Periodo 84
Lunes 8am – 9am	Lunes 9am – 10am		Sábado 9pm – 10pm

}

- ✓ Sea $I = \{i_1, \dots, i_n\}$ el vector de días de la semana que agrupa diferentes periodos. Cada $i \in I$ contiene un conjunto de periodos $T_i \subseteq T$. Dos días distintos no pueden tener periodos iguales, por ejemplo sea T_{i_1} y T_{i_2} dos periodos pertenecientes a los tipos i_1 y i_2 , con $i_1 \neq i_2$ debe ocurrir que $T_{i_1} \cap T_{i_2} = \emptyset$.

Definimos de antemano $I = \{i_l, i_m, i_e, i_j, i_v, i_s\}$, cada elemento correspondiente a los días lunes, martes, miércoles, jueves, viernes y sábado.

Ejemplo: $I = \{$

Lunes	Martes	Miércoles	Jueves	Viernes	Sábado
-------	--------	-----------	--------	---------	--------

}

- ✓ Sea $D = \{1, \dots, d\} \forall d \in Z$ el vector de docentes que pueden dictar un curso cualquiera $c \in C$.

Ejemplo: $D = \{$

Luis Alarcón Loayza	César Alcántara Loayza	...
---------------------	------------------------	-----

}

- ✓ Sea $A = \{1, \dots, a\} \forall a \in Z$ el vector de aulas donde se puede dictar un curso cualquiera $c \in C$.

Ejemplo: $A = \{$

Aula 101	Aula 102	Aula 103	Aula 104	...
----------	----------	----------	----------	-----

}

- ✓ Sea DC la matriz de docentes $d \in D$ que dictan el curso $c \in C$.

Ejemplo: DC= {

	Luis Alarcón Loayza	César Alcántara Loayza	...
Algorítmica I	0	0	
...	0	0	
Base de Datos	1	0	
...	0	0	
Análisis de Sistemas	0	1	
...	0	0	

}

- ✓ Sea TC la matriz de periodos de tiempo $t \in T$ en que se puede dictar el curso $c \in C$.

Ejemplo: TC = {

	Algorítmica I	...	Base de Datos	...	Inteligencia Artificial	...
Periodo 1	1	...	0	...	0	...
Periodo 2	1	...	0	...	0	...
...	1	...	0	...	0	...
Periodo 6	0	...	1	...	0	...
Periodo 7	0	...	1	...	0	...
...	0	...	1	...	0	...
Periodo 11	0	...	0	...	1	...
Periodo 12	0	...	0	...	1	...
...	0	...	0	...	1	...
Periodo 15	1	...	0	...	0	...
...	...	...	...	...	...	...

}

- ✓ Sea TD la matriz de periodos de tiempo $t \in T$ correspondiente a la disponibilidad horaria del docente para dictar un curso $c \in C$.

Ejemplo: $TD = \{$

	Luis Alarcón Loayza	César Augusto Alcántara Loayza	...
Periodo 1	0	0	...
Periodo 2	0	0	...
...	0	0	...
Periodo 6	1	0	...
Periodo 7	1	0	...
...	1	0	...
Periodo 11	0	1	...
Periodo 12	0	1	...
...	0	1	...
Periodo 15	0	0	...
...	...	...	...

$\}$

- ✓ Sea $K = \{k_1, \dots, k_a\} \quad \forall k \in \mathbb{Z}$ el vector de capacidades de las diferentes aulas.

Ejemplo: $K = \{$

50 alumnos (Aula 101)	50	40	30	...
--------------------------	----	----	----	-----

$\}$

- ✓ Sea $Q = \{q_1, \dots, q_c\} \quad \forall q \in \mathbb{Z}$ el vector de cantidad máxima de alumnos por curso.

Ejemplo: $Q = \{$

50 alumnos (Algorítmica I)	50	50	40	...
-------------------------------	----	----	----	-----

$\}$

- ✓ Sea $H = \{h_1, \dots, h_c\} \quad \forall h \in \mathbb{Z}$ el vector de cantidad de horas semanales que se le asigna a cada uno de los cursos.

Ejemplo: $H = \{$

6 horas (Algorítmica I)	5	5	3	...
----------------------------	---	---	---	-----

$\}$

- ✓ Sea $MHD = \{p_1, \dots, p_d\} \forall p \in \mathbb{Z}$ el vector de cantidad máxima de horas que debe dictar semanalmente un docente $d \in D$.

Ejemplo: $MHD = \{$

7 horas (Luis Alarcón Loayza)	8	6	4	...
------------------------------------	---	---	---	-----

$\}$

- ✓ Sea $NHD = \{n_1, \dots, n_d\} \forall n \in \mathbb{Z}$ el vector de cantidad mínima de horas que debe dictar semanalmente un docente $d \in D$.

Ejemplo: $NHD = \{$

5 horas (Luis Alarcón Loayza)	4	3	3	...
------------------------------------	---	---	---	-----

$\}$

- ✓ Sea $S = \{s_1, \dots, s_n\}$ el vector de semestres (o ciclos académicos) que está asociado a los diferentes cursos. Cada $s \in S$ contiene un conjunto de cursos $C_s \subseteq C$, que no pueden ser programadas en un mismo periodo de tiempo. Dos semestres distintos no pueden tener cursos iguales, por ejemplo sea C_{s_1} y C_{s_2} dos cursos pertenecientes a los semestres S_1 y S_2 , con $S_1 \neq S_2$ debe ocurrir que $C_{s_1} \cap C_{s_2} = \emptyset$.

Definimos de antemano $S = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}\}$, cada elemento correspondiente a los cursos de los semestres del 1 al 10.

Ejemplo: $S = \{$

Semestre I	Semestre II	Semestre III	...	Semestre X
---------------	----------------	-----------------	-----	---------------

$\}$

- ✓ Sea $M = \{m_1, \dots, m_n\}$ el vector de tipos para indicar las clases de aulas existentes. Cada $m \in M$ contiene un conjunto de aulas $A_m \subseteq A$. Dos tipos distintos pueden tener aulas iguales, por ejemplo

sea A_{m_1} y A_{m_2} dos aulas pertenecientes a los tipos M_1 y M_2 , con $M_1 \neq M_2$ puede ocurrir que $A_{m_1} \cap A_{m_2} \neq \emptyset$.

Definimos de antemano $M = \{m_t, m_p, m_l\}$, cada elemento correspondiente a las aulas de tipo teoría, práctica y laboratorio.

Ejemplo: $M = \{$

Teoría	Práctica	Laboratorio
--------	----------	-------------

$\}$

- ✓ Sea PrvCD el vector de asignaciones previas de docentes $d \in D$ que dictan el curso $c \in C$.

Ejemplo: PrvCD = {

	Luis Alarcón Loayza	César Alcántara Loayza	...
Algorítmica I	0	0	...
...	0	0	...
Base de Datos	1	0	...
...	0	0	...
Análisis de Sistemas	0	1	...
...	0	0	...
Proyecto de Tesis I	0	1	...
...	...	...	...

}

- ✓ Sea PrvTC el vector de asignaciones previas de un curso $c \in C$ que se dicta en el periodo $t \in T$.

Ejemplo: PrvTC = {

	Algorítmica I	...	Base de Datos	...	Inteligencia Artificial	...
Periodo 1	1	...	0	...	0	...
Periodo 2	0	...	0	...	0	...
...	0	...	0	...	0	...
Periodo 6	0	...	0	...	0	...
Periodo 7	0	...	0	...	0	...
...	0	...	0	...	0	...
Periodo 11	0	...	0	...	1	...
Periodo 12	0	...	1	...	0	...
...	0	...	0	...	0	...

}

- ✓ Sea DTC el vector de periodos deseables $t \in T$ donde un curso $c \in C$ se pueda dictar.

Ejemplo: DTC = {

	Algorítmica I	...	Base de Datos	...	Inteligencia Artificial	...
Periodo 1	1	...	0	...	0	...
Periodo 2	1	...	0	...	0	...
...	0	...	0	...	0	...
Periodo 20	0	...	1	...	0	...
Periodo 21	0	...	1	...	0	...
...	0	...	0	...	0	...
Periodo 39	0	...	0	...	1	...
Periodo 40	0	...	0	...	1	...
...	0	...	0	...	0	...

1.2.3.3.1.2 Parámetros

Definimos los siguientes parámetros que corresponden a cada uno de los recursos involucrados:

y = cantidad de periodos semanales

z = cantidad de aulas,

n = cantidad de cursos,

p = cantidad de docentes,

$$\forall y, z, n, p \in \mathbb{Z}$$

1.2.3.3.1.3 Variables de decisión

Tenemos X_{ijkl} como variables binario-valoradas: 1, si asignan un aula i, al curso j, un docente k, en el periodo l.

$$X_{ijkl} = \begin{cases} 1 & \text{si el aula i es asignada al curso j con el docente k en el periodo l.} \\ 0 & \text{si no.} \end{cases}$$

$$\forall i \in \{1, \dots, z\}, \forall j \in \{1, \dots, n\}, \forall k \in \{1, \dots, p\}, \forall l \in \{1, \dots, y\}$$

1.2.3.3.1.4 Variable de duración

Tenemos W_{ijkl} tiempo de ocupación del aula i por el curso j con el docente k en el periodo l .

$$W_{ijkl} = \begin{cases} R \in Z \\ 0 \end{cases}$$

$$\forall i \in \{1, \dots, z\}, \forall j \in \{1, \dots, n\}, \forall k \in \{1, \dots, p\}, \forall l \in \{1, \dots, y\}$$

1.2.3.3.1.5 Modelamiento de Restricciones Obligatorias

Este modelamiento se dará en función de las restricciones obligatorias mencionadas anteriormente:

Todo curso j que se dicta en un periodo específico l debe tener asignado a lo más un docente k en un aula i .

$$\sum_{i=1}^z \sum_{k=1}^p X_{ijkl} \leq 1 \quad \forall j \in C, \forall l \in T \dots (ROG1)$$

Todo docente k que enseña en un periodo específico l debe tener asignado a lo más un curso j en un aula i .

$$\sum_{i=1}^z \sum_{j=1}^n X_{ijkl} \leq 1 \quad \forall k \in D, \forall l \in T \dots (ROG2)$$

Toda aula i en un periodo específico l debe tener asignado a lo más un curso j y un docente k .

$$\sum_{j=1}^n \sum_{k=1}^p X_{ijkl} \leq 1 \quad \forall i \in A, \forall l \in T \dots (ROG3)$$

Todo docente k debe tener asignado un curso j que corresponda a su preferencia.

[DC = Matriz de docentes y cursos]

$$X_{ijkl} = DC_{jk} \quad \forall X_{ijkl} = 1, \forall i \in A, \forall l \in T, \forall j \in C, \forall k \in D \dots (ROG4)$$

Todo docente k debe tener asignado un curso j dentro de su disponibilidad de tiempo.

[TD = Matriz de docentes y periodos de tiempo]

$$X_{ijkl} = TD_{jk} \quad \forall X_{ijkl} = 1, \forall i \in A, \forall l \in T, \forall j \in C, \forall k \in D \dots (ROG5)$$

Todo curso j debe tener una cantidad de alumnos Q matriculados que no sobrepase la capacidad K del aula i .

$$X_{ijkl} \leq 1 + (K_i - Q_j) \quad \forall X_{ijkl} = 1, \forall i \in A, \forall l \in T, \forall j \in C, \forall k \in D \dots (ROG6)$$

Todo curso j debe cumplir con una cantidad de horas semanales H .

$$\sum_{i=1}^z \sum_{k=1}^p \sum_{l=1}^y X_{ijkl} * W_{ijkl} = H_j \quad \forall j \in C \dots (ROG7.a)$$

Todo curso j cuya cantidad de horas académicas sea par se le asignará bloques de dos horas diarias hasta cubrir la totalidad, sino se le asignará un día de tres horas y el resto de dos horas.

$$W_{ijkl} = 2 + H_j \bmod 2 \quad \forall i \in A, \forall l \in T, \forall j \in C, \forall k \in D \dots (ROG7.b)$$

Todo curso j debe ser únicamente asignado en una aula i con un docente k desde que inicia en un periodo a hasta la finalización de su tiempo de duración W .

$$\sum_{l=a}^{a+W_{jka}} X_{ijkl} \leq 1 \quad \forall X_{ijkl} = 1, \forall i \in A, \forall l, a \in T, \forall j \in C, \forall k \in D \dots (ROG9)$$

Todo docente k tiene una cantidad mínima de horas NHD y una cantidad máxima de horas MHD de dictado de clases. [W = Vector de ocupación del aula i por el curso j con el docente k en el periodo l]

$$\sum_{i=1}^z \sum_{j=1}^p \sum_{l=1}^y X_{ijkl} * W_{ijkl} \leq MHD_k \wedge \sum_{i=1}^z \sum_{j=1}^p \sum_{l=1}^y X_{ijkl} * W_{ijkl} \geq NHD_k \quad \forall k \in D \dots (ROE1)$$

Todo curso j de tipo teoría L_t debe ser asignado a un aula i de tipo no laboratorio M_t .

$$L_{jT} = M_{iT} \quad \forall X_{ijkl} = 1, \forall i \in A, \forall l \in T, \forall j \in C, \forall k \in D \dots (ROE2.a)$$

Todo curso j de tipo práctica L_p debe ser asignado a un aula i de tipo no laboratorio M_p .

$$L_{jP} = M_{iP} \quad \forall X_{ijkl} = 1, \forall i \in A, \forall l \in T, \forall j \in C, \forall k \in D \dots (ROE2.b)$$

Todo curso j de tipo laboratorio LI debe ser asignado a un aula i de tipo laboratorio MI.

$$L_{jL} = M_{iL} \quad \forall X_{ijkl} = 1, \forall i \in A, \forall l \in T, \forall j \in C, \forall k \in D \quad \dots (ROE3)$$

Todo curso j dentro de un rango con valor inicial a y un valor final b que pertenece a un semestre S y a un Grupo G debe ser asignado a un aula i y un docente k y no debe coincidir con otro rango de cursos de otro semestre y con ningún otro grupo.

$$\sum_{k=1}^p \sum_{j=a}^b \sum_{i=1}^z X_{ijkl} \leq 1 \quad \forall l \in T, \forall a, b \in C \wedge \forall [a, b] \subseteq S \wedge \forall [a, b] \subseteq G \dots (ROE4)$$

Existen asignaciones previas de un curso j a un determinado período l y un aula i .

$$\text{PrvCD}_{ijkl} = X_{ijkl} \quad \forall X_{ijkl} = 1, \forall i \in A, \forall l \in T, \forall j \in C, \forall k \in D \dots (ROE5)$$

Existen asignaciones previas de un docente k a un determinado período l y un curso j .

$$\text{PrvTC}_{ijkl} = X_{ijkl} \quad \forall X_{ijkl} = 1, \forall i \in A, \forall l \in T, \forall j \in C, \forall k \in D \dots (ROE6)$$

Los cursos son clasificados en los turnos de mañana, tarde y noche según el semestre al que pertenecen

$$X_{ijkl} = \text{TTC}_{jl} \quad \forall X_{ijkl} = 1, \forall i \in A, \forall l \in T, \forall j \in C, \forall k \in D \dots (ROE7)$$

1.2.3.3.1.6 Modelamiento de Restricciones Deseables

Este modelamiento se dará en función de las restricciones deseables mencionadas anteriormente:

Todo curso j que pertenece a un rango inicial a y rango final b no debe ser asignado en un mismo periodo l con respecto a otro rango inicial c y un rango final d que corresponden a semestres S consecutivos.

$$\sum_{k=1}^p \sum_{j=a}^b \sum_{i=1}^z X_{ijkl} + \sum_{k=1}^p \sum_{j=c}^d \sum_{i=1}^z X_{ijkl} \leq 1 \quad \forall l \in T, \forall a, b, c, d \in C \wedge \forall [a, b], [c, d] \subseteq S \dots (RDG1)$$

Los bloques de un curso deben de distribuirse adecuadamente durante la semana.

$$V_i \leq \text{días que ocupa el curso en la semana} / (\text{ultimo día de la semana que ocupa el curso} - \text{primer día de la semana usado para el curso}) \leq V_s \quad \forall j \in C \dots (RDG2)$$

Minimizar el número de horas libres entre el fin de una clase y el inicio de otra.

$$B_i \leq \text{Total de horas ocupadas por algún curso} / (\text{ultima hora ocupada del día por algún curso} - \text{primera hora ocupada del día por algún curso}) \leq B_s \quad \forall i \in I \dots (RDG3)$$

Todo curso j debe ser dictado en un periodo l determinado.

$$DTC_{jl} = X_{ijkl} \quad \forall X_{ijkl} = 1 \dots (RDG4) \text{ y } (RDE1)$$

Todo curso j debe tener asignado un bloque de horas a lo más una vez dentro de los periodos [a,b] que pertenecen a un día de la semana l.

$$\sum_{i=1}^z \sum_{k=1}^p \sum_{l=a}^b X_{ijkl} \leq 1 \quad \forall j \in C, \forall a, b \in T \wedge \forall [a, b] \subseteq I \dots (RDG5)$$

1.2.3.3.1.7 Función Objetivo

1.2.3.3.1.7.1 Función objetivo global

$$f_{og}(x) = \sum_{x=1}^N f_{ol}(x) \quad \forall x \in \{1, \dots, N\}$$

Donde x representa cada elemento de la matriz, N representa el número total de elementos en la matriz de cuatro dimensiones.

1.2.3.3.1.7.2 Función objetivo local

$$f_{ol}(x) = ((chsl(x) + chsg(x)) / 2) * ((chpl(x) + chpg(x)) / 2)$$

En donde, chsl y chpl representa los conflictos de aulas y docentes en un mismo semestre, respectivamente, mientras

que chsg y chpg representan los conflictos de aulas y docentes entre semestres.

1.2.3.3.1.8 Criterio de Optimización

$Min(fog(x)) = w$, donde w es una constante cualquiera.

1.2.3.3.2. Modelo Particular

Una vez definido el modelo general debemos adaptarlo a la FISI, dando como resultado un modelo particular el cual acota aún más el dominio del problema.

1.2.3.3.2.1 Conjunto de Datos

- ✓ Se cuenta con un total de 68 cursos a programar.
- ✓ Existen a lo máximo 3 grupos por cada curso a programar.
- ✓ Los cursos a programar se pueden dictar en las diferentes modalidades de tipo teoría, practica y laboratorio
- ✓ Existen un total de 84 periodos de tiempo posibles que van desde el lunes a las 8 am. hasta el sábado a las 10:00 pm. definiendo la duración por periodo de 60 minutos.
- ✓ Los diferentes periodos de tiempo son contenidos en los días lunes, martes, miércoles, jueves, viernes y sábado.
- ✓ Se cuenta con un total de una plana de 101 docentes.
- ✓ Se cuenta con un total de 25 aulas hábiles para dictar las clases
- ✓ Son 10 semestres (ciclos académicos) en total.
- ✓ Existen tres clasificaciones diferentes de aulas, de teoría, práctica y laboratorio.

1.2.3.3.2.2 Parámetros

84 = cantidad de periodos semanales

25 = cantidad de aulas,

68 = cantidad de cursos,

101 = cantidad de docentes,

$\forall x, y, z, n, p \in \mathbb{Z}$

1.2.3.3.2.3 Variables de decisión

Tenemos X_{ijkl} como variables binario-valoradas: 1, si asignan una aula i , al curso j , un docente k , en el periodo l

$$X_{ijkl} = \begin{cases} 1 & \text{si el aula } i \text{ es asignada al curso } j \text{ con el docente } k \text{ en el periodo } l. \\ 0 & \text{si no.} \end{cases}$$

$$\forall i \in \{1, \dots, 25\}, \forall j \in \{1, \dots, 68\}, \forall k \in \{1, \dots, 101\}, \forall l \in \{1, \dots, 84\}$$

1.2.3.3.2.4 Variable de duración

Tenemos W_{ijkl} tiempo de ocupación del aula i por el curso j con el docente k en el periodo l .

$$W_{ijkl} = \begin{cases} R \in Z \\ 0 \end{cases}$$

$$\forall i \in \{1, \dots, 25\}, \forall j \in \{1, \dots, 68\}, \forall k \in \{1, \dots, 101\}, \forall l \in \{1, \dots, 84\}$$

1.2.3.3.2.5 Modelamiento de Restricciones Obligatorias

Según las restricciones obligatorias mencionadas anteriormente:

$$\sum_{i=1}^{25} \sum_{k=1}^{101} X_{ijkl} \leq 1 \quad \forall j \in C, \forall l \in T \dots (ROG1)$$

$$\sum_{i=1}^{25} \sum_{j=1}^{68} X_{ijkl} \leq 1 \quad \forall k \in D, \forall l \in T \dots (ROG2)$$

$$\sum_{j=1}^{68} \sum_{k=1}^{101} X_{ijkl} \leq 1 \quad \forall i \in A, \forall l \in T \dots (ROG3)$$

$$X_{ijkl} = DC_{jk} \quad \forall X_{ijkl} = 1, \forall i \in A, \forall l \in T, \forall j \in C, \forall k \in D \dots (ROG4)$$

$$X_{ijkl} = TD_{jk} \quad \forall X_{ijkl} = 1, \forall i \in A, \forall l \in T, \forall j \in C, \forall k \in D \dots (ROG5)$$

$$X_{ijkl} = 1 + (K_i - Q_j) \quad \forall X_{ijkl} = 1, \forall i \in A, \forall l \in T, \forall j \in C, \forall k \in D \dots (ROG6)$$

$$\sum_{i=1}^{25} \sum_{k=1}^{101} \sum_{l=1}^{84} X_{ijkl} * T_{ijkl} = H_j \quad \forall j \in C \dots (ROG7.a)$$

$$T_{ijkl} = 2 + H_j \text{mod} 2 \quad \forall i \in A, \forall l \in T, \forall j \in C, \forall k \in D \dots (ROG7.b)$$

$$\sum_{l=a}^{a+T_{jka}} X_{ijkl} \leq 1 \quad \forall X_{ijka} = 1, \forall i \in A, \forall l \in T, \forall j \in C, \forall k \in D \dots (ROG9)$$

$$\sum_{i=1}^{25} \sum_{j=1}^{68} \sum_{l=1}^{84} X_{ijkl} * W_{ijkl} \leq MHD_k \wedge \sum_{i=1}^{25} \sum_{j=1}^{68} \sum_{l=1}^{84} X_{ijkl} * W_{ijkl} \geq NHD_k \quad \forall k \in D \dots (ROE1)$$

$$L_{iT} = M_{iT} \quad \forall X_{ijkl} = 1, \forall i \in A, \forall l \in T, \forall j \in C, \forall k \in D \dots (ROE2.a)$$

$$L_{jP} = M_{jP} \quad \forall X_{ijkl} = 1, \forall i \in A, \forall l \in T, \forall j \in C, \forall k \in D \dots (ROE2.b)$$

$$L_{jL} = M_{jL} \quad \forall X_{ijkl} = 1, \forall i \in A, \forall l \in T, \forall j \in C, \forall k \in D \dots (ROE3)$$

$$\sum_{i=1}^{25} \sum_{j=a}^b \sum_{l=1}^{101} X_{ijkl} \leq 1 \quad \forall l \in T, \forall a, b \in C \wedge \forall [a, b] \subseteq S \wedge \forall [a, b] \subseteq L \dots (ROE4)$$

$$\text{PrvCD}_{ijkl} = X_{ijkl} \quad \forall X_{ijkl} = 1, \forall i \in A, \forall l \in T, \forall j \in C, \forall k \in D \dots (ROE5)$$

$$\text{PrvTC}_{ijkl} = X_{ijkl} \quad \forall X_{ijkl} = 1, \forall i \in A, \forall l \in T, \forall j \in C, \forall k \in D \dots (ROE6)$$

$$X_{ijkl} = \text{TTC}_{jl} \quad \forall X_{ijkl} = 1, \forall i \in A, \forall l \in T, \forall j \in C, \forall k \in D \dots (ROE7)$$

1.2.3.3.2.6 Modelamiento de Restricciones Deseables

Según las restricciones deseables mencionadas anteriormente:

$$\sum_{i=1}^{25} \sum_{j=a}^b \sum_{k=1}^{101} X_{ijkl} + \sum_{i=1}^{25} \sum_{j=c}^d \sum_{k=1}^{101} X_{ijkl} \leq 1 \quad \forall l \in T, \forall a, b, c, d \in C \wedge \forall [a, b], [c, d] \subseteq S \dots (RDG1)$$

$$Vi \leq \text{días que ocupa el curso en la semana} / (\text{ultimo día de la semana que ocupa el curso} - \text{primer día de la semana usado para el curso}) \leq Vs \quad \forall j \in C \dots (RDG2)$$

$$Bi \leq \text{Total de horas ocupadas por algún curso} / (\text{ultima hora ocupada del día por algún curso} - \text{primera hora ocupada del día por algún curso}) \leq Bs \quad \forall i \in I \dots (RDG3)$$

$$\text{DTC}_{jl} = X_{ijkl} \quad \forall X_{ijkl} = 1 \dots (RDG4) \text{ y } (RDE1)$$

$$\sum_{i=1}^{25} \sum_{k=1}^{101} \sum_{l=a}^b X_{ijkl} \leq 1 \quad \forall j \in C, \forall a, b \in T \wedge \forall [a, b] \subseteq I \dots (RDG5)$$

1.2.3.3.2.7 Función Objetivo

1.2.3.3.2.7.1 Función objetivo global

$$fog(x) = \sum_{x=1}^{14422800} fol(x) \quad \forall x \in \{1, \dots, 14422800\}$$

Donde x representa cada elemento de la matriz, N representa el número total de Elementos en la matriz de cuatro dimensiones. El valor 14422800 es producto de multiplicar las 25 aulas por los 68 cursos por los 101 docentes por los 84 periodos.

1.2.3.3.2.7.2 Función objetivo local

$$fol(x) = ((chsl(x) + chsg(x)) / 2) * ((chpl(x) + chpg(x)) / 2)$$

En donde, chsl y chpl representa los conflictos de aulas y docentes en un mismo semestre, respectivamente, mientras que chsg y chpg representan los conflictos de salones y profesores entre semestres.

1.2.3.3.2.8 Criterio de Optimización

$$Min(fog(x)) = 0$$

1.2.3.4. Complejidad Computacional

Dado un problema de gran escala, como el que estamos tratando, no es suficiente realizar una formulación matemática del problema; debemos también cerciorarnos de que el problema se pueda solucionar dentro de tiempos de procesamiento razonables.

Garey [4] menciona que la teoría de la complejidad computacional, como parte de la teoría de la computación, estudia los recursos requeridos por un algoritmo para resolver un problema. Los recursos más estudiados son el tiempo, que se traduce en el número de pasos de ejecución de un algoritmo para resolver el problema; y el espacio, que es la cantidad de memoria utilizada para resolver el problema.

En este sentido, se pueden clasificar los problemas en tres clases: P, NP y NP-C.

1.2.3.4.1. Problemas P

La clase de los problemas que se pueden resolver mediante algoritmos eficientes se denota como P, indicando tiempo polinomial. Esta definición permite observar que la mayor parte de los problemas tratables tienen soluciones prácticas.

Tales problemas son de hecho increíbles, puesto que significa que podemos formular problemas de tamaño enorme sin tener que preocuparnos de que el problema se pueda solucionar dentro del tiempo que dure nuestra vida, es decir, sin limitaciones de tiempo. Los programas lineales con variables continuas son uno de los tantos problemas polinómicos y éste es la razón por la que no son alarmantes los problemas que puedan tener una gran cantidad de variables.

1.2.3.4.2. Problemas NP

La clase de problemas para las que existe un algoritmo no determinista y cuyo tiempo de ejecución es polinomial respecto al tamaño de los datos de entrada se llama NP, estos problemas se procesan en un tiempo bastante comparable a los polinómicos cuando el problema es pequeño, pero su estructura de tiempo de procesamiento hace imposible solucionarlo cuando el tamaño del problema es grande. La importancia de esta clase de problemas es que contiene muchos problemas de búsqueda y de optimización para los que se desea saber si existe alguna solución o si existe una mejor solución que las conocidas.

Dada su importancia, se han hecho muchos esfuerzos para encontrar algoritmos que decidan algún problema de NP en tiempo polinomial. Sin embargo, pareciera que para algunos problemas de NP, los del conjunto NP-completo no es posible encontrar un algoritmo mejor que simplemente realizar una búsqueda exhaustiva.

1.2.3.4.3. Problemas NP-Completo

La clase de complejidad NP-Completo o polinómicos totalmente no determinista es el subconjunto de los problemas en NP, tal que, todo problema en NP se puede reducir en cada uno de los problemas NP-Completo. Se puede decir que esta clase de problemas son los más difíciles en NP y muy probablemente no formen parte de la clase de complejidad P [3]. La razón es que de tenerse una solución polinomial para un problema de NP-Completo, todos los problemas de NP tendrían también una solución en tiempo polinomial.

[3] menciona que un algoritmo de fuerza bruta para problemas NP-Completo utiliza tiempo exponencial con respecto al tamaño de la entrada.

Se desconoce si hay mejores algoritmos, por lo cual Cormen [14] señala que para resolver un problema NP-Completo de tamaño arbitrario se utiliza uno de los siguientes enfoques:

- ✓ Aproximación: Un algoritmo que rápidamente encuentra una solución no necesariamente óptima, pero dentro de un cierto rango de error. En algunos casos, encontrar una buena aproximación es suficiente para resolver el problema, pero no todos los problemas NP-Completo tienen buenos algoritmos de aproximación.
- ✓ Probabilidad: Un algoritmo probabilístico obtiene en promedio una buena solución al problema planteado, para una distribución de los datos de entrada dada.
- ✓ Heurísticas: Un algoritmo que trabaja razonablemente bien en muchos casos. En general son rápidos, pero no existe medida de la calidad de la respuesta.

Ahora volviendo a nuestro problema, desafortunadamente, si encontramos su número cromático, este caerá en los problemas NP-completo. Algunos de estos problemas NP-completo son la programación entera, el problema de la mochila, el problema del coloreado de grafos, el problema del vendedor viajero, el problema del ciclo hamiltoniano, etc. Sabemos que un problema es NP-completo si podemos demostrar que el problema se puede exponer

en forma modificada como un problema NP-completo conocido. Es decir, el isomorfismo a un cierto problema NP-completo conocido puede ser demostrado.

Para ilustrar nuestro punto de vista, consideremos un problema de tamaño N , donde las funciones siguientes representan el tiempo de procesamiento en microsegundos, y además se examinará en la siguiente tabla el comportamiento de cada tipo de funciones, a medida que el tiempo N vaya aumentando:

Tiempo de la Función	Problema de tamaño N					
	10	20	30	40	50	60
N	.00001 segundos	.00002 segundos	.00003 segundos	.00004 segundos	.00005 segundos	.00006 segundos
N^2	.0001 segundos	.0004 segundos	.0009 segundos	.0016 segundos	.0025 segundos	.0036 segundos
N^5	1 segundos	3.2 segundos	24.3 segundos	1.7 minutos	5.2 minutos	13.0 minutos
2^N	.001 segundos	1.0 segundos	17.9 minutos	12.7 días	35.7 años	366 siglos

Tabla 1.1 Tiempos de los algoritmos

Como se muestra en la Tabla 1.1, las funciones polinómicas N , N^2 , y N^5 requieren tiempos de cómputo de segundos en problemas pequeños y de minutos en problemas grandes. Sin embargo, los tiempos de los problemas no-polinómicos crecen a partir de segundos a días e incluso a siglos, cuando el tamaño del problema aumenta. Es decir, el tiempo de procesamiento incrementa mientras que el problema se hace más grande.

¿Bueno, pero las computadoras cada día son mas rápidas? Hay puntos de vista que están en conflicto con la posibilidad de aumentar la velocidad de procesamiento de nuestras computadoras. Por un lado, algunos la ven como finita puesto que los chips de silicio se calientan demasiado debido a la miniaturización de los circuitos que son requeridos para que funcione la computadora a las enormes velocidades de reloj. Otros son más optimistas y abrazan la ley de Moore, que postula que la velocidad se continuará duplicando cada año. Tomando el caso más optimista, la velocidad de cómputo continuará aumentando, después de 10 generaciones de evolución de la computadora. Tendremos máquinas que sean 1000 veces más rápidas que las computadoras que existen actualmente y los siglos del tiempo de cómputo serán reducidos a ¡todavía en años de la magnitud! Esperanzadamente, la velocidad en que se procesa podrá mejorar para tratar más rápidamente soluciones a los problemas NP-completos.

Como podemos notar, no es factible solucionar los problemas NP-completos, sin modificaciones adaptantes. Pero dado que, igualmente, se tiene que dar una solución a los problemas NP-completos, el acercamiento logrado por las heurísticas llega a ser una salida viable. La búsqueda heurística no nos proporciona una solución exacta, pero en el mejor de los casos nos brinda una solución aproximada, y por lo tanto no tiene definido un grado óptimo global. Esto podría llegar a ser tedioso, ya que no sabríamos si existe una solución mejor o no. También, grandes esfuerzos se han realizado para cerciorarse de que la heurística se establezca y converja en un resultado deseado. No obstante, dada la imposibilidad de solucionar problemas exponenciales y NP-completos, la búsqueda heurística aproxima suficientemente y eficientemente un problema NP-completo, entonces éste será un gran intento de obtener algunas soluciones satisfactorias dentro de tiempos razonables, con respecto a la obtención de soluciones con tiempos mayores al transcurso de nuestra vida.

1.3 Delimitaciones de la investigación

1.3.1. Delimitación Espacial

La presente investigación tiene como ámbito geográfico la Facultad de Ingeniería de Sistemas e Informática de la Universidad Nacional Mayor de San Marcos.

1.3.2. Delimitación Social

Esta investigación delimita su estudio al beneficio que puedan obtener las autoridades encargadas de la elaboración de horarios de clases en cuanto a la agilización de sus procesos, además de la satisfacción generada para el alumnado y los docentes.

1.3.3. Delimitación Técnica

Esta investigación delimita su estudio técnico al desarrollo del análisis y diseño de un sistema de apoyo a la decisión basado en algoritmos genéticos que ayude a la elaboración de horarios de clases.

1.4 Justificación

Con este desarrollo se pretende obtener un modelo cuyo principal objetivo es el de reducir el tiempo y la ocurrencia de errores en la generación manual de los horarios de clase por parte de las autoridades responsables de la Facultad de Ingeniería de Sistemas e Informática de la Universidad Nacional Mayor de San Marcos, y satisfacer al máximo las necesidades de los usuarios. Esto significa un avance tecnológico y organizativo cuyo propósito es el de minimizar los cruces entre cursos y horas, evitando extensos intervalos entre clases y disminuyendo los niveles de insatisfacción entre los docentes y estudiantes.

1.4.1. Justificación de carácter práctico

Desde el punto de vista práctico se plantea diseñar un sistema de apoyo a la decisión que a partir de las entradas genere un conjunto de posibles

horarios factibles de los cuales uno, previa selección por parte de las autoridades, podría convertirse en un horario de clase definitivo para su uso en el semestre en curso.

1.4.2. Justificación de carácter metodológico

Desde el punto de vista metodológico esta investigación es importante porque plantea una nueva variante del algoritmo genético básico aplicado a la generación de horarios de clases a nivel universitario además de aplicarlo en cada una de las fases del problema.

1.5 Objetivos

1.5.1. Objetivo General

El objetivo general de nuestro trabajo es automatizar el proceso de generación de horarios de clases con un margen de error de 2%, mediante el uso de algoritmos genéticos, con el fin de reducir el tiempo y los recursos empleados en su elaboración, además de aumentar la calidad de los mismos y elevar el grado de satisfacción por parte del alumnado y el docente.

1.6 Hipótesis General

La automatización del proceso de generación de horarios de clases con un margen de error de 2%, mediante el uso de algoritmos genéticos, reducirá el tiempo y los recursos empleados, además de aumentar la calidad de los mismos y elevar el grado de satisfacción por parte del alumnado y el docente.

1.7 Variables e Indicadores

A continuación detallaremos la variable independiente y las variables dependientes que forman parte de la hipótesis planteada, así como también el indicador asociado a dichas variables:

Variable independiente 1: La automatización del proceso de generación de horarios de clases.

→ Indicador

- Número de tareas manuales realizadas en el proceso de generación de horarios.

Variable Dependiente 1: Margen de error del 2%.

→ Indicador

- Numero de clases del horario académico semestral que infrinjan una o más restricciones del problema.

Variable Dependiente 2: Reducir el tiempo de generación de horarios.

→ Indicador

- Tiempo en la generación de horarios de clases.

Variable Dependiente 3: Reducir el uso de recursos implicados en la elaboración de los horarios de clases

→ Indicador

- Costo de los recursos involucrados en la elaboración de horarios de clases.

Variable Dependiente 4: Aumentar la calidad de la solución.

→ Indicador

- Numero de clases del horario académico semestral que infrinjan una o más restricciones del problema.

Variable Dependiente 5: Aumentar la satisfacción del alumnado.

→ Indicador

- Número de reclamos presentados por los alumnos.

Variable Dependiente 6: Aumentar la satisfacción del docente

→ Indicador

- Número de reclamos presentados por los docentes.

Capítulo 2: Marco Teórico

En este capítulo se presentarán los conceptos básicos que se requieren para abordar los algoritmos genéticos, así como un ejemplo que permita comprender cómo aplicarlo a un problema de elección.

2.1. El proceso de evolución

La mayoría de los organismos evolucionan a través de dos procesos primarios: la selección natural y la reproducción. Mediante la selección se determina qué miembros de una población sobreviven para reproducirse, y mediante la reproducción se asegura la recombinación de los genes de la descendencia. Esta mezcla del material genético permite que las especies evolucionen mucho más rápidamente de lo que lo harían si tuvieran sólo la copia de los genes de uno de sus progenitores.

El principio fundamental de la selección natural fue formulado por Darwin, anteriormente al descubrimiento de los mecanismos genéticos, al observar que una diferencia entre el crecimiento de una población y los recursos disponibles llevaría a situaciones de competición y adaptación. Desconocedor en aquel momento de los

principios básicos de la herencia mendeliana, Darwin conjeturó la fusión de las cualidades de los padres mezclándose en los organismos de sus descendientes.

La teoría de la evolución defendida por Charles Darwin se sustenta en los siguientes argumentos [18]:

- La naturaleza está en constante evolución por lo tanto las especies cambian continuamente; con el paso del tiempo algunas se extinguen y otras nuevas aparecen.
- El proceso de cambio es gradual y continuo, no se produce a saltos discontinuos o por transformaciones súbitas.
- Los organismos que presentan semejanzas están emparentados.
- El cambio evolutivo es el resultado del proceso de selección natural: los individuos más eficientes a la hora de proporcionarse alimentos y hábitat, y con más aptitudes para dejar un mayor número de descendientes, son los más favorecidos y transmiten a sus hijos los caracteres favorables. En la lucha por la existencia sobreviven los más aptos.

Las cuestiones, relacionadas con los mecanismos hereditarios, a las que la teoría de Darwin no podía responder han sido contestadas con el desarrollo de la Genética.

Mendel descubrió los principios básicos de la transferencia de factores hereditarios de padres a hijos, donde se estableció que los cromosomas son los principales portadores de la información hereditaria y que los genes, que representan los factores hereditarios, están alineados en cromosomas. El origen de las variaciones en la herencia se explica por la existencia de ciertos cambios (mutaciones) en el texto genético, pudiendo sufrir mutación todos los organismos de forma aleatoria. Pero además, en aquellos individuos que se reproducen sexualmente se puede considerar el proceso por el cual las características de los organismos se mezclan al combinar su ADN (cruce). Este proceso es la fuente de inspiración de los algoritmos genéticos y, en consecuencia, su idea básica es imitar lo que hace la naturaleza. Por esta razón estos algoritmos utilizan un vocabulario tomado de la Genética.

2.2. Introducción a los algoritmos genéticos

Los primeros intentos serios de relacionar la informática y la evolución surgieron a principios de los años sesenta cuando varios biólogos comenzaron a experimentar con simulaciones de sistemas genéticos; esto es, modelos computacionales que imitan la evolución biológica.

En los últimos años, se ha mostrado un creciente interés en una nueva técnica de búsqueda basada en la teoría de la evolución y que se conoce como **Algoritmo Genético**. Esta técnica se basa en los mecanismos de selección que utiliza la naturaleza, de acuerdo a los cuales los individuos más aptos de una población son los que sobreviven, al adaptarse más fácilmente a los cambios que se producen en su entorno.

La técnica de los Algoritmos Genéticos se encuentra dentro de un marco más amplio que viene a ser la Inteligencia Artificial, y más aún en la rama que se ha denominado Computación Evolutiva.

El término Computación Evolutiva se refiere al estudio de los fundamentos y aplicaciones de ciertas técnicas heurísticas de búsqueda basadas en los principios naturales de la evolución. Una gran variedad de algoritmos evolutivos han sido propuestos, pero principalmente pueden clasificarse en: Algoritmos Genéticos, Programación Evolutiva, Estrategias Evolutivas, Sistemas Clasificadores y Programación Genética. Esta clasificación se basa sobre todo en detalles de desarrollo histórico más que en el hecho de un funcionamiento realmente diferente, de hecho las bases biológicas en las que se apoyan son esencialmente las mismas. Las diferencias entre ellos se centran en los operadores que se usan en cada caso y en general en la forma de implementar la selección, reproducción y sustitución de individuos en una población [33].

Aunque los detalles de la evolución no han sido completamente comprendidos, incluso hoy en día, existen algunos puntos en los que se fundamentan [33]:

- La evolución es un proceso que opera a nivel de cromosomas, y no a nivel de individuos. Cada individuo es codificado como un conjunto de cromosomas.
- La selección natural es el mecanismo mediante el cual los individuos mejor adaptados son los que tienen mayores posibilidades de reproducirse.
- El proceso evolutivo tiene lugar en la etapa de la reproducción. Es en esta

etapa donde se produce la mutación, que es la causante de que los cromosomas de los hijos puedan ser diferentes a los de los padres, y el cruce, que combina los cromosomas de los padres para que los hijos tengan cromosomas diferentes.

De forma breve, comentaremos cada uno de los algoritmos evolutivos mencionados anteriormente, para tener una idea de las similitudes y diferencias entre ellos [33]:

- Los Algoritmos Genéticos resuelven los problemas generando poblaciones sucesivas a las que se aplican los operadores de mutación y cruce. Cada individuo representa una solución al problema, y se trata de encontrar al individuo que represente a la mejor solución.
- La Programación Genética funciona igual que la técnica anterior pero se centra en el estudio de problemas cuya solución es un programa. De manera que los individuos de la población son programas que se acercan más o menos a realizar una tarea que es la solución.
- La Programación Evolutiva es otro enfoque de los algoritmos genéticos, en este caso el estudio se centra en conseguir operadores genéticos que imiten lo mejor posible a la naturaleza, en cada caso, más que en la relación de los padres con su descendencia. En este caso no se utiliza el operador de cruce, tomando la máxima importancia el operador de mutación.
- Estrategias Evolutivas se centran en el estudio de problemas de optimización e incluyen una visión del aprendizaje en dos niveles: a nivel de genotipo, y a nivel de fenotipo.
- Y por último los Sistemas Clasificadores engloban el estudio de problemas en los que la solución buscada se corresponde con toda una población.

El comienzo del desarrollo de los algoritmos genéticos se ha debido realmente al trabajo de John Holland, investigador matemático de la Universidad de Michigan, quien estaba convencido de que la recombinación de grupos de genes, que se realiza mediante el cruce, era la parte más importante de la evolución. Desde pequeño se preguntaba cómo logra la naturaleza crear seres cada vez más perfectos (aunque, esto no es totalmente cierto, o en todo caso depende de qué entienda uno por perfecto). Lo curioso era que todo se lleva a cabo a base de interacciones locales entre individuos, y entre éstos y lo que les rodea.

Fue a través de la lectura de un libro escrito por un biólogo evolucionista, R. A. Fisher, titulado "La teoría genética de la selección natural", como comenzó a

descubrir los medios de llevar a cabo sus propósitos de comprensión de la naturaleza. De ese libro aprendió que la evolución era una forma de adaptación más potente que el simple aprendizaje. Su objetivo era lograr que las computadoras aprendieran por sí mismas. A la técnica que inventó Holland se le llamó originalmente "planes reproductivos", pero fue a principios de los 60, en la Universidad de Michigan, donde Holland impartía un curso titulado Teoría de Sistemas Adaptativos y con la participación de sus alumnos crearon las ideas que más tarde se convertirían en una técnica de programación, el **Algoritmo Genético**, que se adapta a la evolución tanto por el cruce como por la mutación. Durante la década siguiente trabajó para ampliar el alcance de este tipo de algoritmos y fruto de ese trabajo publica en 1975 la primera monografía sobre el tema, *Adaptation in Natural and Artificial Systems*, en la que se sientan las bases teóricas que fundamentan el desarrollo de los Algoritmos Genéticos desde el punto de vista computacional, abstrae los conceptos de la genética natural y los aplica a la economía y al reconocimiento de patrones.

El propósito de Holland era estudiar de un modo formal el fenómeno de la adaptación, tal y como ocurre en la naturaleza, y desarrollar vías de extrapolar esos mecanismos de adaptación natural a los sistemas computacionales.

Desde entonces el campo de aplicaciones de este tipo de algoritmos no ha dejado de crecer: diseños de turbinas de aviones, predicciones de la evolución bursátil, estudios de progresión de enfermedades (mediante comparación de imágenes tomadas por resonancia magnética), análisis de la estructura del cristal líquido, son algunos de los ejemplos.

2.3. Definición de algoritmos genéticos

Una definición bastante completa de algoritmo genético es la propuesta por John Koza [12]:

"Es un algoritmo matemático altamente paralelo que transforma un conjunto de objetos matemáticos individuales con respecto al tiempo usando operaciones modeladas de acuerdo al principio Darwiniano de reproducción y supervivencia del más apto, y tras haberse presentado de forma natural una serie de operaciones genéticas de entre las que destaca la recombinación sexual. Cada uno de estos objetos matemáticos suele ser una cadena de caracteres (letras o números) de longitud fija que se ajusta al modelo de las cadenas de cromosomas, y se les asocia con una cierta función matemática que refleja su aptitud. "

Según Michalewicz [18]: “En general, cualquier tarea abstracta para ser llevada a cabo puede ser pensada como la solución de un problema. Para los espacios pequeños, los métodos clásicos exhaustivos son por lo general suficientes; para grandes espacios deben ser empleadas técnicas especiales de inteligencia artificial. Los algoritmos genéticos están entre esas técnicas, son algoritmos estocásticos cuyos métodos de búsqueda modelan algunos fenómenos naturales: la herencia genética y la lucha darwiniana por la supervivencia.”

“Los Algoritmos Genéticos (AG) son métodos adaptativos que pueden usarse para resolver problemas de búsqueda y optimización, y están basados en el proceso genético de los organismos vivos. A lo largo de las generaciones, las poblaciones evolucionan en la naturaleza de acorde con los principios de la selección natural y la supervivencia de los más fuertes. Por imitación de este proceso, los Algoritmos Genéticos son capaces de ir creando soluciones para problemas del mundo real. La evolución de dichas soluciones hacia valores óptimos del problema depende en buena medida de una adecuada codificación de las mismas y del diseño del algoritmo genético” [5].

Un algoritmo genético consiste en una función matemática o una rutina de software que toma como entradas a los individuos de una población y retorna como salidas cuales de ellos deben generar descendencia para la nueva generación. Dado un problema específico a resolver, la entrada del Algoritmo Genético es un conjunto de soluciones potenciales a ese problema, codificadas de alguna manera, y una métrica llamada función de aptitud que permite evaluar cuantitativamente a cada individuo. Estos individuos pueden ser soluciones que se sabe que funcionan, con el objetivo de que el Algoritmo Genético las mejore, pero que se suelen generar aleatoriamente.

Un punto clave de este modelo, es que el proceso de adaptación no se hace cambiando incrementalmente un solo individuo, sino manteniendo una población de individuos a partir de los cuales se generan nuevos individuos haciendo uso de los operadores genéticos. Cada individuo en la población está asociado con una aptitud y los valores se usan en competencia para determinar qué individuos serán usados para formar nuevos individuos. Los mejores individuos o soluciones sobreviven a lo largo de las generaciones y las peores van desapareciendo y son sustituidas por nuevas.

Una de las características de los AG es su habilidad de explotar información acumulada acerca de un espacio de búsqueda inicialmente desconocido para guiar la búsqueda subsecuente a subespacios útiles. Su aplicación está enfocada sobre todo a espacios de búsqueda grandes, complejos y poco entendidos.

2.4. Analogía de un algoritmo genético con la naturaleza

En la naturaleza, los individuos compiten entre si por recursos tales como comida, agua y refugio. Además, los animales de la misma especie normalmente antagonizan para obtener una pareja. Aquellos individuos que tengan más éxito tendrán probablemente un número mayor de descendientes, por lo tanto, mayores probabilidades de que sus genes sean propagados a lo largo de sucesivas generaciones. La combinación de características de los padres bien adaptados, en un descendiente, puede producir muchas veces un nuevo individuo mucho mejor adaptado que cualquiera de sus padres a las características de su medio ambiente [9].

Los Algoritmos Genéticos utilizan una analogía directa del fenómeno de evolución en la naturaleza. Trabajan con una población de individuos, cada uno representando una posible solución a un problema dado. Además recibe una calificación relacionada con la eficiencia de cada posible solución, siguiendo con la relación análoga esto vendría siendo en la naturaleza la capacidad de supervivencia de cada individuo o la capacidad de competencia con otros individuos para obtener recursos. Con respecto a esto, a cada individuo se le asigna una puntuación de adaptación dependiendo de que tan buena sea la respuesta al problema. A los más adaptados se les da la oportunidad de reproducirse mediante cruzamientos con otros individuos de la población, produciendo descendientes con características de ambos padres. Al producirse estas reproducciones se da paso a una nueva generación que a su vez se convierte en una nueva población. Los miembros menos adaptados poseen pocas probabilidades de que sean seleccionados para la reproducción, y desaparecen. Una nueva población de posibles soluciones es generada mediante la selección de los mejores individuos de la generación actual, emparejándolos entre ellos para producir un nuevo conjunto de individuos. Esta nueva generación contiene una proporción más alta de las características poseídas por los mejores miembros de la generación anterior. De esta forma, a lo largo de varias generaciones, las características buenas son difundidas a lo largo de la población mezclándose con otras. Con la nueva

población se repite el proceso hasta poder encontrar los resultados esperados en un individuo o en una población. Favoreciendo el emparejamiento de los individuos mejor adaptados, es posible recorrer las áreas más prometedoras del espacio de búsqueda [9].

Si el Algoritmo Genético ha sido diseñado correctamente, la población convergerá a una solución óptima o aceptable al problema. Los dos procesos que más contribuyen a la evolución son el cruzamiento y la adaptación basada en la selección y reproducción. La mutación también juega un papel significativo, pero no se puede determinar qué tan importante sea su rol; además no debe ser utilizada demasiado ya que el Algoritmo Genético se puede convertir en una búsqueda al azar, pero su utilización asegura que ningún punto en el espacio de búsqueda tenga probabilidad cero de ser examinado.

Se han realizado diversos estudios en profundidad acerca de los mecanismos de la evolución, y aunque quedan partes por entender, muchos aspectos están bastante explicados. De manera muy general podemos decir que, en la evolución de los seres vivos el problema al que cada individuo se enfrenta cada día es la supervivencia. Para ello cuenta con las habilidades innatas provistas en su material genético. A nivel de los genes, el problema es el de buscar aquellas adaptaciones beneficiosas en un medio hostil y cambiante. Debido en parte a la selección natural, cada especie gana una cierta cantidad de "conocimiento", el cual es incorporado a la información de sus cromosomas.

Así pues, la evolución tiene lugar en los cromosomas, en donde está codificada toda la información del ser vivo. La información almacenada en el cromosoma varía de unas generaciones a otras. En el proceso de formación de un nuevo individuo, se combina la información cromosómica de los progenitores aunque la forma exacta en que se realiza es desconocida.

Aunque muchos aspectos no estén totalmente resueltos, podemos definir algunos principios generales aceptados. Entre ellos tenemos:

1. La evolución se realiza a nivel de los cromosomas en lugar de los individuos a los que representan.
2. La selección natural es el proceso por el que los cromosomas con "buenas estructuras" se reproducen más a menudo que los demás.

3. La evolución tiene lugar en el proceso de reproducción mediante la combinación de los cromosomas de los progenitores. Llamamos recombinación al proceso que tiene como resultado la formación del cromosoma del descendiente. También son de tener en cuenta las mutaciones que pueden alterar dichos códigos.
4. La evolución biológica no posee un almacenamiento en memoria, en el sentido de que durante la formación de los cromosomas solamente se considera la información del período anterior.

En el contexto de los algoritmos genéticos, la idea básica es la siguiente: generar un conjunto con algunas de las posibles soluciones. Cada una va a ser llamada **individuo**, y a dicho conjunto se le denominará **población**. Un individuo se entiende como una solución posible al problema que puede representarse mediante una cadena de cromosomas. Se denomina **cromosoma** a cada una de las cadenas de ADN que se encuentran en el núcleo de las células y son responsables de la transmisión de información genética. Hoy en día se sabe que los cambios de dicha información se efectúan en los **genes** de un individuo (unidad básica de codificación de cada uno de los atributos de un ser vivo), y que sus atributos más deseables, por ejemplo los que le permiten adaptarse mejor a su entorno, se transmiten a sus descendientes cuando éste se reproduce sexualmente. Un gen es fundamentalmente una unidad de herencia, una sección de ADN que codifica una cierta función bioquímica. Dependiendo de su especie, un organismo puede tener un número variable de genes en su ADN (desde una docena de genes hasta decenas de miles). Cada gen es capaz de ocupar una única región en particular de un cromosoma, a lo cual se le llama **locus**. A su vez pueden existir formas alternativas o valores del gen llamadas **alelos**.

Las características de los individuos, sean beneficiosas o no, se van a denominar **fenotipos**. Un fenotipo puede estar en más de un cromosoma, en cuyo caso puede ser que el hijo herede un fenotipo que no tenía ni el padre ni la madre, sino una combinación de ambos. Un ejemplo en el humano es el color de la piel o la estructura del cráneo. A su vez se denomina **genotipo** a la información contenida en el genoma de un individuo. Puede verse como lo que potencialmente puede llegar a ser el individuo. Por lo tanto el **fenotipo** podría entenderse como una solución de-codificada mientras que el **genotipo** como una solución codificada.

Los rasgos específicos y observables de un individuo constituyen su fenotipo. En caso de que el hijo tenga parte de los genes del padre y parte de los genes de la

madre que intervienen en un fenotipo, se va a crear una característica nueva asociada a ese fenotipo. De todas formas, no es un enfoque muy frecuente, ya que debemos asegurar que el conjunto de los fenomas tendrá ley de composición interna respecto al operador de cruce definido sobre el **alfabeto cromosómico**, el cual es utilizado para representar los valores de cada gen de los cromosomas del individuo. Por otro lado, que un cromosoma codifique más de un fenotipo es más raro todavía. El cromosoma debe tener en dicho caso tantos valores como el producto del número de valores posibles que tenga cada fenotipo del cromosoma. Es de vital importancia la forma de codificar los fenotipos en los cromosomas y la determinación de qué es fenotipo; es decir, cómo la información va a ser almacenada en el código genético. Escoger equivocadamente la forma de almacenar la información puede ralentizar la convergencia, es decir, que se tardará más en encontrar la solución o nunca convergerá en una solución debido a que la población está errando aleatoriamente por efecto de las mutaciones y de los cruzamientos sin llegar nunca a un punto estable, a este fenómeno se le denomina deriva genética.

Dado que un algoritmo genético actúa como un método de búsqueda multidireccional sobre una población de posibles soluciones, ésta sufre la simulación de una evolución: En cada generación las soluciones relativamente buenas se reproducen, mientras las relativamente malas mueren. Durante la evaluación se decodifica el gen y se convierte en una serie de parámetros de un problema. Luego se halla la solución del problema a partir de esos parámetros y se le da una puntuación en función de lo cerca que esté de la mejor solución. A esta puntuación se le llama **fitness**. Para distinguir las diferentes soluciones se usa una función evaluadora que juega el papel del entorno. Esta función se denomina **función de adaptación**, función de ajuste o función de idoneidad lo cual determina el grado de adaptación de un individuo y, en el campo de la optimización funcional, es la función objetivo del problema en cuestión; es decir, la función a optimizar. La **selección** es el proceso mediante el cual algunos individuos en una población son seleccionados para reproducirse basados en su aptitud. Tenemos la “selección dura” que se da cuando solamente los mejores individuos se mantienen para generar progenia y la “selección blanda” cuando se utilizan mecanismos probabilísticos para mantener como padres a individuos que tengan aptitudes relativamente bajas. Posteriormente se establece un proceso de creación de un nuevo individuo a partir de dos progenitores (reproducción sexual) o de un único progenitor (reproducción asexual) a lo cual se le denomina **reproducción**. Durante

la reproducción sexual ocurre la **recombinación** o **cruzamiento**. La presión de selección o presión selectiva determina la intensidad con la que el entorno tiende a dar ventajas adaptativas a organismos o a eliminar su información genética. La **mutación** consiste en el cambio individual de nucleótidos de padre a hijo. En general, las mutaciones son provocadas por errores probabilísticos en el mecanismo de replicación del ADN.

2.5. Anatomía de un algoritmo genético

Cuando las personas se enfrentaron con problemas que no podían ser solucionados por métodos matemáticos o analíticos, y que la única forma de resolverlos era a través de prueba y error dirigido, es decir, probar dónde se crea que va a mejorar el resultado, se dieron cuenta de que este proceso era similar al proceso que seguía la naturaleza, así que se intentó copiar su manera de operar y se crearon los algoritmos genéticos que imitan el proceso evolutivo de la naturaleza.

Un algoritmo genético está compuesto por [22]:

- ✓ **Módulo evolutivo:** Presenta un mecanismo de decodificación que se encarga de interpretar la información de un individuo y una función de evaluación que mide la calidad del mismo.
- ✓ **Módulo poblacional:** Posee una representación poblacional y las técnicas necesarias para poder manipularla como son la técnica de representación, el criterio de selección y de reemplazo. Aquí también se define el tamaño de la población y la condición de terminación.
- ✓ **Módulo reproductivo:** Contiene los operadores genéticos.

2.6. Componentes de un algoritmo genético

A grandes rasgos, un algoritmo genético hace lo siguiente: genera una población inicial de soluciones, calcula su rendimiento, selecciona padres y reproduce soluciones utilizando información de los padres; este proceso se repite hasta que se cumpla una condición de terminación previamente establecida.

Un Algoritmo genético tiene cinco componentes básicos:

- Una **representación** de las soluciones potenciales del problema.

Antes de que un algoritmo genético pueda empezar a trabajar en un problema, se necesita un método para codificar las soluciones potenciales del problema de forma que una computadora pueda procesarlas. Un enfoque común es codificar las soluciones como cadenas binarias: secuencias de 1s y 0s, donde el dígito de cada posición representa el valor de algún aspecto de la solución. Otro método similar consiste en codificar las soluciones como cadenas de enteros o números decimales, donde cada posición, de nuevo, representa algún aspecto particular de la solución. Este método permite una mayor precisión y complejidad que el método comparativamente restringido de utilizar sólo números binarios, y a menudo está intuitivamente más cerca del espacio de problemas. Un tercer método consiste en representar a los individuos de un AG como cadenas de letras, donde cada letra representa un aspecto específico de la solución. La virtud de estos tres métodos es que facilitan la definición de operadores que causen los cambios aleatorios en los individuos seleccionados: cambiar un 0 por un 1 o viceversa, sumar o restar al valor de un número una cantidad elegida al azar, o cambiar una letra por otra.

- Un procedimiento para crear una **población inicial** de posibles soluciones. La población inicial suele ser generada aleatoriamente, pero dependiendo del problema, también se pueden utilizar otros métodos para obtener esta solución inicial. Últimamente se están utilizando métodos heurísticos para generar soluciones iniciales de buena calidad. En este caso, es importante garantizar la diversidad estructural de estas soluciones para tener una representación de la mayor parte de población posible o al menos evitar la convergencia prematura.
- Una **función de evaluación** que permite clasificar las soluciones en términos de su aptitud.

En general, los problemas de optimización tienen una función que se quiere minimizar o maximizar. Cada vez que se obtiene una solución cualquiera, ésta tiene un valor de dicha función, con este valor se pueden comparar las soluciones y determinar cuál es mejor. A esta función se le llama función objetivo. Dicha función guía el mecanismo de búsqueda a través del operador de selección. La influencia de la función es fundamental para determinar los individuos candidatos a sobrevivir que serán aquellos individuos a los que se aplicarán los operadores evolutivos.

- Un conjunto de **operadores de evolución** que alteran la composición de los individuos de la población a través de las generaciones.

Una vez que se tienen seleccionados los padres es necesario combinarlos para generar nuevos cromosomas que nos sirvan para ir mejorando los rendimientos y a partir de ahí llegar a una solución final. Una manera de hacer esta reproducción es tomar información de cada uno de los padres y combinarla, es decir, se seleccionan algunos genes de los padres y se mezclan para crear un nuevo cromosoma. A este proceso se le determina recombinación (cruzamiento o emparejamiento). También se pueden reproducir los cromosomas mediante un proceso llamado mutación, por medio del cual se toma un cromosoma, y se modifican algunos de sus genes, de acuerdo a un porcentaje llamado porcentaje de mutación, para producir un nuevo cromosoma. Dependiendo de cada las características del problema que se quiere resolver se pueden utilizar diferentes métodos para la reproducción de cromosomas, incluso se pueden combinar métodos. En algunas ocasiones parte de la población es reproducida por medio de recombinación y el resto por medio de mutación

- Una **configuración paramétrica** de elementos tales como el tamaño de la población, probabilidad de cruzamiento, probabilidad de mutación, criterio de parada, etc.

Sobre la configuración paramétrica podemos indicar lo siguiente:

- En general los AG trabajan sobre una población fija de n individuos, inicializada aleatoriamente.
- El tamaño de la población debe ser un número tal que permita mantener diversidad en los individuos solución, sin sacrificar la eficiencia computacional del mecanismo de búsqueda. El tamaño de esta población depende de varios factores, uno de estos es el tipo de representación que se tiene para los cromosomas. Usualmente, con la representación binaria, las poblaciones suelen ser grandes mientras que con otro tipo de representaciones se reduce el tamaño de la población.
- Las probabilidades de aplicación de los operadores evolutivos definen el balance entre la exploración del espacio de búsqueda y la explotación de buenas soluciones.

- El criterio de parada puede especificar un esfuerzo prefijado (en número de generaciones) o involucrar aspectos dinámicos de la evolución (variaciones en los valores – mejor o promedio) de la función de fitness.
- Usualmente se determinan los valores apropiados mediante análisis empíricos.

2.7. Características de los algoritmos genéticos

Para resolver un problema mediante algoritmos genéticos se deben considerar las siguientes características [18]:

- Son algoritmos estocásticos. Dos ejecuciones distintas pueden dar dos soluciones distintas. Esto es útil por el hecho de que hay gran cantidad de soluciones válidas, por lo que es interesante que distintas ejecuciones nos puedan dar resultados distintos.
- Son algoritmos de búsqueda múltiple, de los que se obtiene como respuesta varias soluciones. Aunque habitualmente los individuos de la población final son similares, los individuos suelen ser distintos entre sí. Con el modelo de paralelización, la probabilidad de obtener muchas soluciones distintas es más alta todavía; por ello, nos podemos quedar con la solución que más nos convenga según la naturaleza del problema.
- Son los algoritmos que hacen una barrida mayor al subespacio de posibles soluciones válidas. De hecho, se considera que, de todos los algoritmos de optimización estocásticos, los algoritmos genéticos son de los más exploratorios disponibles.
- A diferencia de otros algoritmos, cuya convergencia y resultado final son fuertemente dependientes de la posición inicial, en los algoritmos genéticos - salvo poblaciones iniciales realmente degeneradas, en los que el operador de mutación va a tener mucho trabajo - la convergencia del algoritmo es poco sensible a la población inicial si ésta se escoge de forma aleatoria y es lo suficientemente grande.

El algoritmo genético es excelente descartando subespacios realmente malos. Cada cierto tiempo, la población vuelve dar el salto evolutivo, y se produce un incremento en la velocidad de convergencia excepcional. La razón de esto es que algunas veces aparece una mutación altamente beneficiosa, o un individuo excepcional, que propaga algún conjunto de cromosomas excepcional al resto de la población.

- La optimización es función de la representación de los datos. Este es el concepto clave dentro de los algoritmos genéticos, ya que una buena codificación puede hacer la programación y la resolución muy sencillas, mientras que una codificación errada nos va a obligar a estudiar que el nuevo genoma cumple las restricciones del problema, y en muchos problemas tendremos que abortar los que no cumplan las restricciones, por ser éstas demasiado complejas. Además, la velocidad de convergencia va a estar fuertemente influenciada por la representación.
- Es una búsqueda paramétricamente robusta. Eso quiere decir que hemos de escoger adecuadamente los parámetros del algoritmo para que converja. Con tasas razonables, va a converger -mejor o peor- en una solución razonablemente buena si la representación es la adecuada.
- Por último, los algoritmos genéticos son intrínsecamente paralelos. Esto significa que, independientemente de que se implemente de forma paralela o no, buscan en distintos puntos del espacio de soluciones de forma paralela. Ese paralelismo intrínseco permite que sean fácilmente paralelizables, es decir, que sea fácil modificar el código para que se ejecute simultáneamente en varios procesadores.

2.8. Algoritmo genético simple

Existen múltiples propuestas y variantes de algoritmos genéticos. En esta parte estudiaremos la propuesta original de Goldberg (1989), conocida como ALGORITMO GENÉTICO SIMPLE.

La representación tradicionalmente utilizada es una cadena binaria. A la cadena general se le llama cromosoma. A cada subcadena (posición en la cadena general) se le denomina gen y al valor dentro de esta posición se le llama alelo. La longitud de la representación depende de las características del problema (número de variables, número de funciones objetivo, dimensión del dominio), y de características de la solución buscada (precisión deseada, por ejemplo).

La representación es el genotipo que se corresponde con una solución al problema (fenotipo). Existe un proceso de codificación (y su inverso de decodificación) que permite pasar de fenotipo a genotipo y viceversa.

La codificación especifica una función de correspondencia $fC: S \rightarrow \{0,1\}^*$, siendo S el espacio de soluciones del problema. La función inversa es la decodificación $fD: \{0,1\}^* \rightarrow S$, puede ser una función parcial. La complejidad de fC y de fD dependerá de las características del problema y de las variables a codificar.

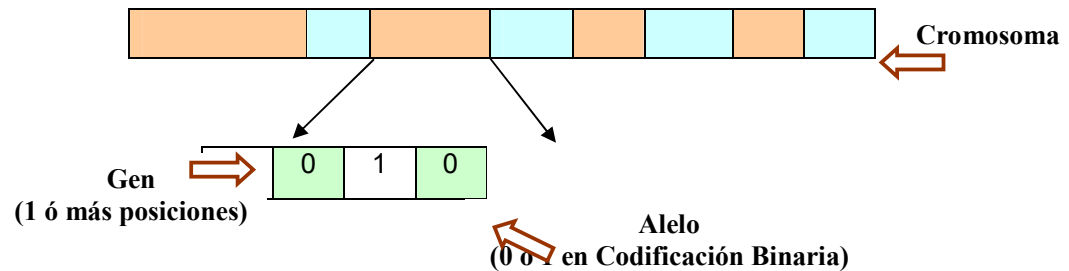


Figura 2.1. Individuo genético binario

2.8.1. Funcionamiento

El proceso algorítmico básico del AG es como sigue:

1. [Inicio] se genera la población aleatoria de n cromosomas (soluciones posibles para el problema).
2. [Aptitud] se evalúa la aptitud $f(x)$ de cada cromosoma x de la población.
3. [Prueba] si la condición de término está satisfecha, se para el algoritmo, se devuelve la mejor solución de la población actual y se va al paso 7.
4. [Nueva población] se crea una nueva población repitiendo los siguientes pasos, hasta que se cumpla la condición de parada.
 - a. [Selección] se selecciona dos cromosomas padres, de una población, según su aptitud (cuanto mejor es la aptitud, mayor es la probabilidad de ser seleccionado).
 - b. [Emparejamiento] con una probabilidad de emparejamiento, los padres se emparejan para formar a un nuevo descendiente (hijos). Si no se realiza emparejamiento alguno, el descendiente es la copia exacta de los padres.
 - c. [Mutación] con una probabilidad de mutación, el nuevo descendiente muta (en alguna posición de su cromosoma).

5. [Sustituir] la nueva población generada es aplicada para otra iteración del algoritmo.
6. [Bucle] se va al paso 2.
7. Fin del algoritmo.

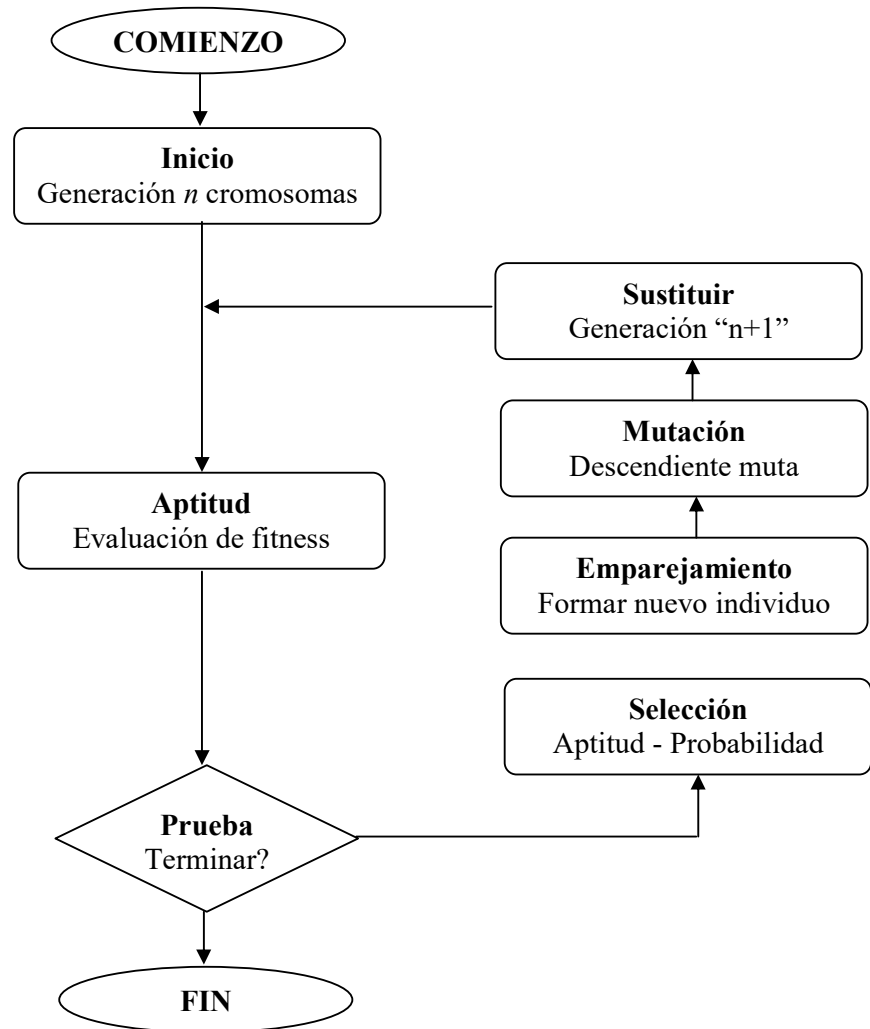


Figura 2.2. Funcionamiento de un algoritmo genético

2.8.2. Algoritmo básico

El Algoritmo Genético Simple, también denominado Canónico, presenta un conjunto de pasos u operaciones que permiten hallar la solución a un problema. Dado un estado inicial y una entrada, a través de pasos sucesivos y bien definidos se llega a un estado final, obteniendo una solución.

A continuación mostraremos el procedimiento básico del AG:

```
Begin  
    t := 0  
    Inicializar P(t) // Generar una población inicial  
    Evaluar P(t) // Computar la función de evaluación  
    Mientras no sea condición de término, repetir  
        t := t+1  
        Seleccionar P(t+1) a partir de P(t)  
        Emparejamiento sobre P(t) // Obtener descendientes  
        Aplicar Mutación sobre P(t) // Según probabilidad  
        Sustituir P(t) por P(t+1) // Insertar la nueva generación  
        Evaluar P(t) // Computar la función de evaluación  
    Fin Mientras  
End
```

Figura 2.3. Pseudocódigo de un algoritmo genético

Donde $P(t)$ es la población de individuos en la generación t ; Emparejamiento y Mutación son operadores genéticos que permiten recombinar la información contenida en los cromosomas. En la aplicación de algoritmos genéticos, se necesita una codificación o representación del problema que resulte adecuada al mismo. Además se requiere una función de evaluación o de adaptación al problema, la cual asigna un valor a cada posible solución codificada indicando la bondad de la solución. Durante la ejecución del algoritmo, los padres deben ser seleccionados para la reproducción o intercambio genético lo que producirá nuevos hijos o soluciones, a los cuales, con cierta probabilidad, se les aplicará una mutación. El resultado de la combinación de los pasos anteriores será un conjunto de individuos, posibles soluciones al problema, los cuales pasarán a formar parte de la siguiente generación.

2.9. Codificación de variables

Si un problema puede ser representado por un conjunto de parámetros (conocidos como genes), éstos pueden ser unidos para formar una cadena de valores (cromosoma); a éste proceso se le llama codificación.

Existen varios aspectos relacionados con la codificación de un problema que deben ser tomados en cuenta en el momento de su realización:

- Se debe utilizar el alfabeto más pequeño posible para representar los parámetros, normalmente se utilizan dígitos binarios
- Las variables que representan los parámetros del problema deben ser discretizadas para poder representarse con cadenas de bits, hay que utilizar suficiente resolución para asegurar que la salida tiene un nivel de precisión adecuado. Se asume que la discretización es representativa de la función objetivo.
- La mayor parte de los problemas tratados con Algoritmos Genéticos son no lineales y muchas veces existen relaciones “ocultas” entre las variables que conforman la solución. Esta interacción es referida como epístasis, y es necesario tomar la en cuenta para una representación adecuada del problema.
- El tratamiento de los genotipos inválidos debe ser tomado en cuenta para el diseño de la codificación. Supóngase que se necesitan 1200 valores para representar una variable, esto requiere al menos 11 bits, pero estos codifican un total de 2048 posibilidades, "sobrando" 848 patrones de bits no necesarios. A estos patrones se les puede dar un valor cero de adaptación, ser substituidos por un valor real, o eliminar el cromosoma.

Antes de que un algoritmo genético pueda empezar a trabajar en un problema, se necesita un método para codificar las soluciones potenciales del problema de forma que una computadora pueda procesarlas.

Los tres métodos principales de codificación de variables son:

- Cadenas binarias; un enfoque común es codificar las soluciones como cadenas binarias, es decir, secuencias de 1's y 0's, donde el dígito de cada posición representa el valor de algún aspecto de la solución.

- Cadenas de enteros o números decimales; otro método similar consiste en codificar las soluciones como cadenas de enteros o números decimales, donde cada posición representa algún aspecto particular de la solución. Este método permite una mayor precisión y complejidad que el método comparativamente restringido de utilizar sólo números binarios, y a menudo “está intuitivamente más cerca del espacio de problemas”.
- Cadenas de letras; un tercer método consiste en representar a los individuos de un AG como cadenas de letras, donde cada letra representa un aspecto específico de la solución.

La codificación realizada a través de cadenas binarias es un tipo de codificación indirecta, mientras que la de cadenas de enteros y letras son un tipo de codificación directa.

La virtud de estos tres métodos es que facilitan la definición de operadores que causen los cambios aleatorios en los individuos seleccionados: cambiar un 0 por un 1 o viceversa, sumar o restar al valor de un número una cantidad elegida al azar, o cambiar una letra por otra.

La elección de la codificación de las variables no es trivial, especialmente si estamos tratando un problema de optimización. Por ejemplo una codificación directa de los números enteros puede dar problemas a la hora de que el algoritmo converja, ya que números consecutivos, como por ejemplo el número 15 y el 16, al pasarlos a binario son muy diferentes ($16 = 10000$ y $15 = 01111$) con lo que una solución con el valor 15 en un campo, difícilmente llegará a evolucionar a una solución con el valor 16 (ya que debería cambiar simultáneamente todos los bits). Este problema es conocido como "Picos de Hamming".

2.10. Parámetros de Configuración

Un algoritmo genético tiene una serie de parámetros que se tienen que fijar para cada ejecución que se realice, los cuales son:

2.10.1. Condición de Término

El Algoritmo Genético se deberá detener cuando se alcance la solución óptima, pero ésta generalmente se desconoce, por lo que se

deben utilizar otros criterios de detención. Normalmente se usan dos criterios: correr el algoritmo genético un número máximo de iteraciones (generaciones) o detenerlo cuando no haya cambios en la población.

2.10.2. Tamaño de la Población

Debe ser suficiente para garantizar la diversidad de las soluciones, y, además, tiene que ver más o menos con el número de bits del cromosoma, aunque nadie ha aclarado cómo tiene que hacerlo. Por supuesto, depende también de la computadora en la que se esté ejecutando.

2.10.3. Porcentaje de Emparejamiento

Indica con qué frecuencia se cruzarán los individuos. Si éste es 0%, los hijos serán como los padres y sólo serán alterados por la mutación. Si éste es 100% todos los individuos nuevos serán creados mediante emparejamiento de los padres de la generación previa. Cuanto más se emparejen los individuos se supone que los hijos serán mejores; sin embargo, es recomendable, por la naturaleza del algoritmo genético, que algunos individuos pasen sin modificarse a la siguiente generación.

2.10.4. Porcentaje de Mutación

Establece la probabilidad con la cual los individuos serán mutados. Si éste porcentaje es 0% los individuos generados después de aplicarse el emparejamiento no sufrirán ningún cambio por el contrario si es de 100% todos los individuos de la población sufrirán cambios o mutaciones. La mutación trata de impedir que la búsqueda del algoritmo genético caiga en extremos locales por eso es conveniente que ocurra de vez en cuando. No es bueno, sin embargo, que la mutación ocurra continuamente, ya que la búsqueda del algoritmo genético pasa de ser “inteligente” a búsqueda aleatoria.

2.11. Operadores Genéticos

Los operadores genéticos consisten en una serie de pasos estructurados que se aplican a los cromosomas y forman resultados concretos, al igual que las operaciones matemáticas y lógicas. Son los diferentes métodos u operaciones que se pueden ejecutar sobre una población y se dividen en 4 categorías: Selección, Emparejamiento, Mutación y Reemplazo.

- **Selección:** Proceso que escoge los miembros de la población que serán utilizados para la reproducción. Su meta es dar más oportunidades de selección a los miembros más aptos de la población.
- **Emparejamiento:** Consiste en unir de alguna forma los cromosomas de dos padres para formar dos descendientes. Existen diversas variaciones, dependiendo del número de puntos de división a emplear, la forma de ver el cromosoma, etc.
- **Mutación:** Se encarga de modificar en forma aleatoria uno o más genes del cromosoma de un descendiente.
- **Reemplazo:** Es el método por el cual se insertan los hijos en la población; por ejemplo, mediante la eliminación del individuo más débil o al azar.

2.11.1. Selección

La selección de los padres viene dada habitualmente mediante probabilidades según su fitness. Un algoritmo genético puede utilizar muchas técnicas diferentes para seleccionar a los individuos que deben copiarse hacia la siguiente generación. Algunos de estos métodos son mutuamente exclusivos, pero otros pueden utilizarse en combinación, algo que se hace a menudo.

2.11.1.1. Elitista

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

2.11.1.2. Proporcional a la aptitud

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

2.11.1.3. Escalada

Al incrementarse la aptitud media de la población, la fuerza de la presión selectiva también aumenta y la función de aptitud se hace más discriminadora. Este método puede ser útil para seleccionar más tarde, cuando todos los individuos tengan una aptitud relativamente alta y sólo se distinguen por pequeñas diferencias en la aptitud.

2.11.1.4. Basado en 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.

2.11.1.5. Rueda de 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. Conceptualmente, esto puede representarse como un juego de ruleta, donde cada individuo obtiene una sección de la ruleta, pero los más aptos obtienen secciones mayores que las de los menos aptos. Luego la ruleta se hace girar, y en cada vez se elige al individuo que “posea” la sección en la que se pare la ruleta.

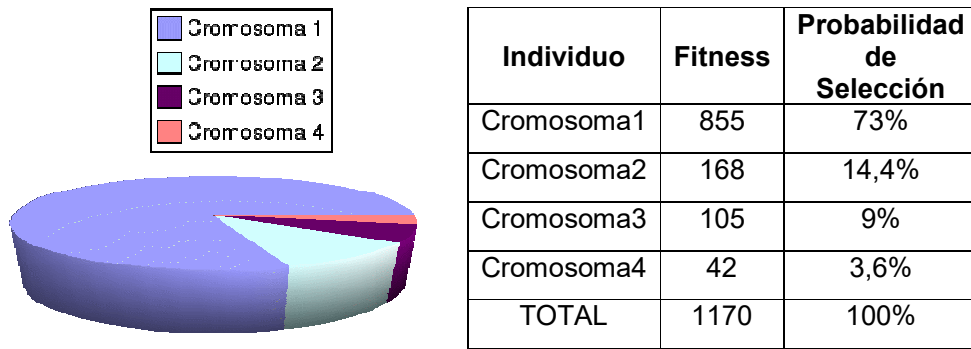


Figura 2.4. Construcción de Ruleta con probabilidades

2.11.1.6. Por Torneo

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.

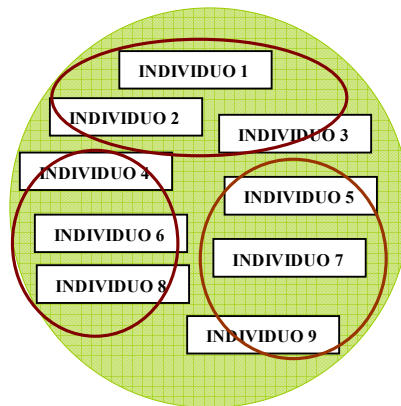


Figura 2.5. Esquematzación de Selección por Torneo

2.11.1.7. Por Estado Estacionario

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

2.11.1.8. Generacional

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

2.11.1.9. 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 pocos 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.

2.11.2. Emparejamiento

Se denomina operador de emparejamiento a la forma de calcular el genoma del nuevo individuo en función de los genomas padres. El operador de emparejamiento es fuertemente responsable de las propiedades del algoritmo genético, y determinará en gran medida la evolución de la población. El proceso de emparejamiento toma dos individuos y corta sus cromosomas en una posición seleccionada al azar, para producir dos segmentos anteriores y dos posteriores, los posteriores se intercambian para obtener dos cromosomas nuevos.

Existe gran cantidad de técnicas de emparejamiento. Las técnicas básicas son las siguientes:

2.11.2.1. Emparejamiento 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 emparejamiento de un punto o de dos puntos.

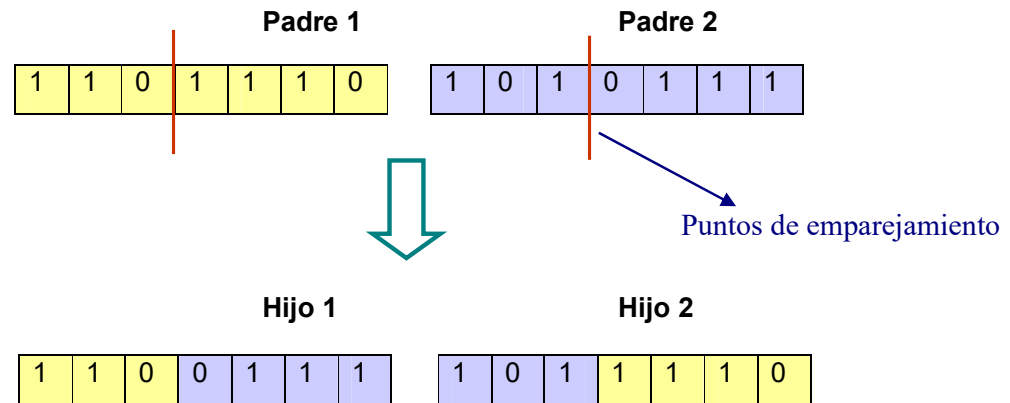


Figura 2.6. Emparejamiento de 1 punto

2.11.2.2. Emparejamiento uniforme

Se genera un patrón aleatorio de 1's y 0's, 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 1 y 0, y si supera una determinada probabilidad se intercambia ese 1 y 0 entre los dos cromosomas.

2.11.2.3. Emparejamiento segmentado

Existe una probabilidad de que un cromosoma sea punto de un emparejamiento. Conforme se va formando la nueva cadena del descendiente, para cada gen, se verifica si ahí se va producir un emparejamiento.

2.11.2.4. Emparejamientos Especializados

En algunos problemas, aplicar aleatoriamente el emparejamiento da lugar a cromosomas que codifican soluciones inválidas; en este caso hay que aplicar el emparejamiento de forma que genere siempre soluciones válidas

2.11.3. Mutación

La operación de mutación más sencilla, y una de las más utilizadas consiste en reemplazar con cierta probabilidad el valor de un bit. Notar que el papel que juega la mutación es el de introducir un factor de diversificación ya que, en ocasiones, la convergencia del

procedimiento a buenas soluciones puede ser prematura y quedarse atrapado en óptimos locales. Otra forma obvia de introducir nuevos elementos en una población es recombinar elementos tomados al azar sin considerar su fitness.

La mutación es aplicada a cada descendiente individualmente luego de cada emparejamiento. Altera un gen al azar, con una probabilidad muy pequeña (típicamente 0.001).

Existen varias técnicas distintas de mutación. Algunas de éstas son:

2.11.3.1. Mutación de bit

Existe una única probabilidad de que se produzca una mutación de algún bit. De producirse, el algoritmo toma aleatoriamente un bit, y lo invierte.

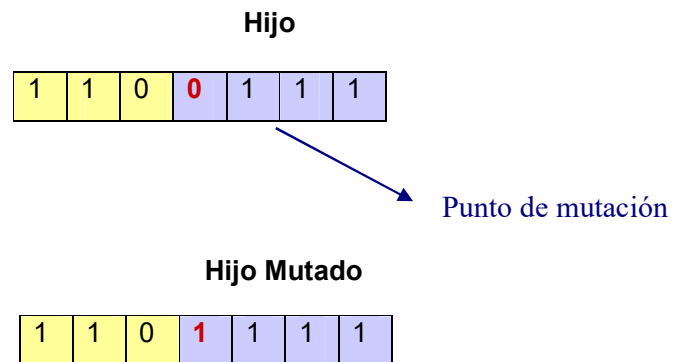


Figura 2.7. Mutación de un bit

2.11.3.2. Mutación multibit

Cada bit tiene una probabilidad de mutarse o no, que es calculada en cada pasada del operador de mutación multibit.

2.11.3.3. Mutación de gen

Igual que la mutación de bit, solamente que, en vez de cambiar un bit, cambia un gen completo. Puede sumar un valor aleatorio, un valor constante, o introducir un gen aleatorio nuevo.

2.11.3.4. Mutación multigen

Igual que la mutación de multibit, solamente que, en vez de cambiar un conjunto de bits, cambia un conjunto de genes. Puede sumar un

valor aleatorio, un valor constante, o introducir un gen aleatorio nuevo.

2.11.3.5. Mutación de intercambio

Existe una probabilidad de que se produzca una mutación. De producirse, toma dos bits/genos aleatoriamente y los intercambia.

2.11.3.6. Mutación de barajado

Existe una probabilidad de que se produzca una mutación. De producirse, toma dos bits o dos genes aleatoriamente y baraja de forma aleatoria los bits o genes, según hubiéramos escogido-comprendidos entre los dos.

Algunas de las razones que pueden motivar a incorporar la mutación son:

- Desbloqueo del algoritmo. Si el algoritmo se bloqueó en un mínimo local, una mutación puede llevarnos a un genotipo mejor dotado.
- Acabar con poblaciones degeneradas. Puede ocurrir que, bien por haber un cuasi-mínimo, ya sea porque en pasos iniciales apareció un individuo demasiado bueno que acabó con la diversidad genética, la población tenga los mismos fenotipos. Si se ha llegado a una población degenerada, es preciso que las mutaciones introduzcan nuevos genomas.
- Incrementar el número de saltos evolutivos. Los saltos evolutivos -aparición de un fenotipo especialmente valioso, o dicho de otra forma, salida de un mínimo local - son muy poco probables en un genético puro para un problema genérico. La mutación permite explorar nuevos subespacios de soluciones, por lo que, si el subespacio es bueno en términos de adaptación, se producirá un salto evolutivo después de la mutación que se expandirá de forma exponencial por la población.
- Enriquecer la diversidad genética. Es un caso más suave que el de una población degenerada, es decir, que la población tenga una diversidad genética pobre; en este caso la mutación

es un mecanismo de prevención de las poblaciones degeneradas.

Sin embargo, si la tasa de mutación es excesivamente alta tendremos la ya conocida deriva genética. Una estrategia muy empleada es una tasa de mutación alta al inicio del algoritmo, para aumentar la diversidad genética, y una tasa de mutación baja al final del algoritmo, para conseguir que converja.

2.12. Ventajas y desventajas

Entre las ventajas que presentan los algoritmos genéticos tenemos:

1. Lo más importante de utilizar AG es que no es necesario tener un conocimiento profundo del problema que se desea resolver, sino que se parte de simples y pequeñas estructuras que interactúan entre si, dejando que sea el proceso de evolución quien haga el correspondiente trabajo. Un AG es un método de programación opuesto al habitual: en vez de descomponer un problema en subproblemas, creamos los subproblemas más sencillos que podamos imaginar y dejamos que se combinen entre sí. Únicamente basta con identificar cualitativamente en qué casos las cadenas se acercan o se alejan de la solución buscada, para que el sistema se perfeccione automáticamente.
2. Los AG son adaptativos, ya que son capaces de solucionar problemas que varían en el tiempo. La gran sencillez de los AG los hace muy atractivos para la programación, pero existen casos en los que puede ser difícil, imposible o poco práctico aplicarlos.
3. Nos permite poder trabajar sobre múltiples soluciones. Cuando se tienen problemas con espacios de soluciones de gran tamaño, y no es posible utilizar enumeración exhaustiva (analizar una por una todas las soluciones posibles) como método de búsqueda, es necesario buscar maneras alternativas de búsqueda para una buena solución (aunque no sea la mejor). Operan en forma simultánea con varias soluciones, en vez de trabajar secuencialmente como las técnicas tradicionales.

4. Cuando se utilizan para problemas de optimización como maximizar una función objetivo, resultan menos afectados por los máximos locales (falsas soluciones) que las técnicas tradicionales.
5. Posee eficiencia en el cálculo de una solución objetivo y resulta fácil su ejecución. Además resultan fáciles de ejecutar en arquitecturas modernas con paralelismo masivo.
6. Usan operadores probabilísticos, en vez de los típicos operadores determinísticos de las otras técnicas.

Además de las ventajas descritas, los AG presentan algunas desventajas tales como:

1. Pueden tardar mucho en alcanzar la convergencia, o no llegar a converger en absoluto, dependiendo en cierta medida de los parámetros que se utilicen, tales como el tamaño de la población, el número de generaciones, etc.
2. Pueden llegar a una convergencia prematura debido a un conjunto de problemas de diversa índole, como mala elección de los operadores genéticos o los porcentajes probables de emparejamiento y/o mutación, entre otros.

2.13. Cómo saber si es posible usar un Algoritmo Genético

La aplicación más común de los algoritmos genéticos ha sido la solución de problemas de optimización, en donde han mostrado ser muy eficientes y confiables; sin embargo, no todos los problemas pueden ser apropiados para la técnica, es por ello que se recomienda, en general, tomar en cuenta las siguientes características del mismo antes de intentar usarla [5]:

- Su espacio de búsqueda que corresponde a sus posibles soluciones, debe estar delimitado dentro de un cierto rango. Este primer punto es muy importante y lo más recomendable, ya que se debe intentar resolver problemas que tengan espacios de búsqueda discretos aunque éstos sean muy grandes. Sin embargo, también podrá intentarse el uso de la técnica

con espacios de búsqueda continuos, pero preferentemente cuando exista un rango de soluciones relativamente pequeño.

- Debe poderse definir una función de aptitud que nos indique qué tan buena o mala es una determinada respuesta.
- Las soluciones deben codificarse de una forma que resulte relativamente fácil de implementar en la computadora.

2.14. Ejemplo del funcionamiento de un algoritmo genético

Con la finalidad de aclarar mejor los conceptos descritos previamente, presentamos ahora una sencilla aplicación del algoritmo genético a un problema de optimización tomado de [11]:

2.14.1. Un Problema de ejemplo

Un grupo de financieros mexicanos ha resuelto invertir 10 millones de pesos en la nueva marca de vino "Carta Nueva". Así pues, en 4 ciudades de las principales de México se decide iniciar una vigorosa campaña comercial:

México en el centro, Monterrey en el noroeste, Guadalajara en el occidente y Veracruz en el oriente. A esas 4 ciudades van a corresponder las zonas comerciales I, II, III y IV. Un estudio de mercado ha sido realizado en cada una de las zonas citadas y han sido establecidas curvas de ganancias medias en función de las inversiones totales (almacenes, tiendas de venta, representantes, publicidad, etc.). Estos datos se ilustran en la Tabla 2.1 y en la Figura 2.9.

Para simplificar los cálculos, supondremos que las asignaciones de créditos o de inversiones deben hacerse por unidades de 1 millón de pesos. La pregunta es: ¿En dónde se deben de asignar los 10 millones de pesos de los que se dispone para que la ganancia total sea máxima?

2.14.2. Representación genética del ejemplo

Lo primero que necesitamos determinar para poder aplicar el algoritmo genético, es cuál será el esquema a utilizarse para

representar las posibles soluciones del problema. En este caso necesitamos 4 bits ($2^4 = 16$) para representar cada solución, porque cada una admite 11 valores posibles (de 0 a 10). Como existen 4 valores independientes (uno por cada zona de estudio), se requieren entonces 16 bits (4×4) por cada cromosoma. De tal forma, una cadena representativa de un cromosoma será como se muestra en la Figura 2.8. Es importante hacer notar que se requiere una función de codificación (esto es, que transforme el valor de la inversión a binario) y una de decodificación (esto es, que realice el proceso inverso). Debido a que en este caso los 4 bits utilizados para representar una solución pueden producir más valores de los que se necesitan, se usará una función de ajuste que haga que los resultados producidos siempre se encuentren en el rango válido.

Inversión (en millones)	Beneficio I	Beneficio II	Beneficio III	Beneficio IV
0	0	0	0	0
1	0.28	0.25	0.15	0.20
2	0.45	0.41	0.25	0.33
3	0.65	0.55	0.40	0.42
4	0.78	0.65	0.50	0.48
5	0.90	0.75	0.62	0.53
6	1.02	0.80	0.73	0.56
7	1.13	0.85	0.82	0.58
8	1.23	0.88	0.90	0.60
9	1.32	0.90	0.96	0.60
10	1.38	0.90	1.00	0.60

Tabla 2.1 Datos obtenidos con la investigación de mercado en cada una de las regiones en estudio

2.14.3. Función de aptitud del ejemplo

Dado que el objetivo es obtener las inversiones que sumen 10, y que tengan un beneficio máximo, podemos usar la siguiente función de aptitud penalizada:

$$F(x) = \frac{c1 + c2 + c3 + c4}{500 * v + 1}$$

Donde $c1$, $c2$, $c3$ y $c4$ son las ganancias por zona, que se calculan de acuerdo a los valores de la Tabla 2.1, y v es el valor absoluto de

la diferencia entre la suma obtenida y 10. Nótese que cuando no se viole ninguna restricción (es decir, cuando la suma de inversiones sea exactamente 10) la función de aptitud no será "castigada".

Las cadenas 1, 2, 3 y 4 corresponden a las cantidades invertidas en las zonas económicas respectivas.

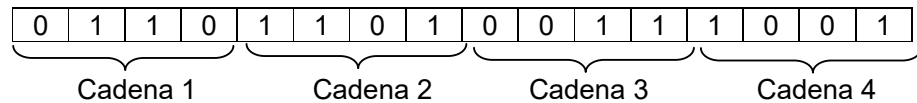


Figura 2.8. Cadena representativa de un cromosoma del ejemplo

2.14.4. Operadores del ejemplo

Se usará un emparejamiento de 2 puntos y la probabilidad que se dará a la misma será del 80%. En cuanto a la mutación, se le asignará una probabilidad baja, tal y como sugiere Goldberg [9], por lo que será del orden del 1%. El tamaño de población manejado para este ejemplo será de 50 cromosomas, y se correrá el algoritmo genético durante 20 generaciones.

2.14.5. Resultados del ejemplo

El resultado obtenido en una corrida típica es de \$1.81 (en millones de pesos), correspondiente a los valores de $C1 = 4$ millones, $C2 = 3$ millones, $C3 = 1$ millón y $C4 = 2$ millones. Esta es la solución óptima, la cual se obtuvo originalmente mediante programación dinámica [12]. El tiempo que le tomó al algoritmo genético encontrar este valor fue de sólo 13 segundos. Las pruebas se corrieron en una computadora IBM PC Compatible 486DX/2 de 66 MHz con coprocesador matemático. Debe hacerse notar que, en este caso, si deseáramos analizar inversiones que sumen otra cantidad, y en unidades menores al millón, el algoritmo genético tendría que modificarse de manera mínima, mientras que la programación dinámica requeriría una cantidad tal de trabajo que prácticamente se volvería inoperante.

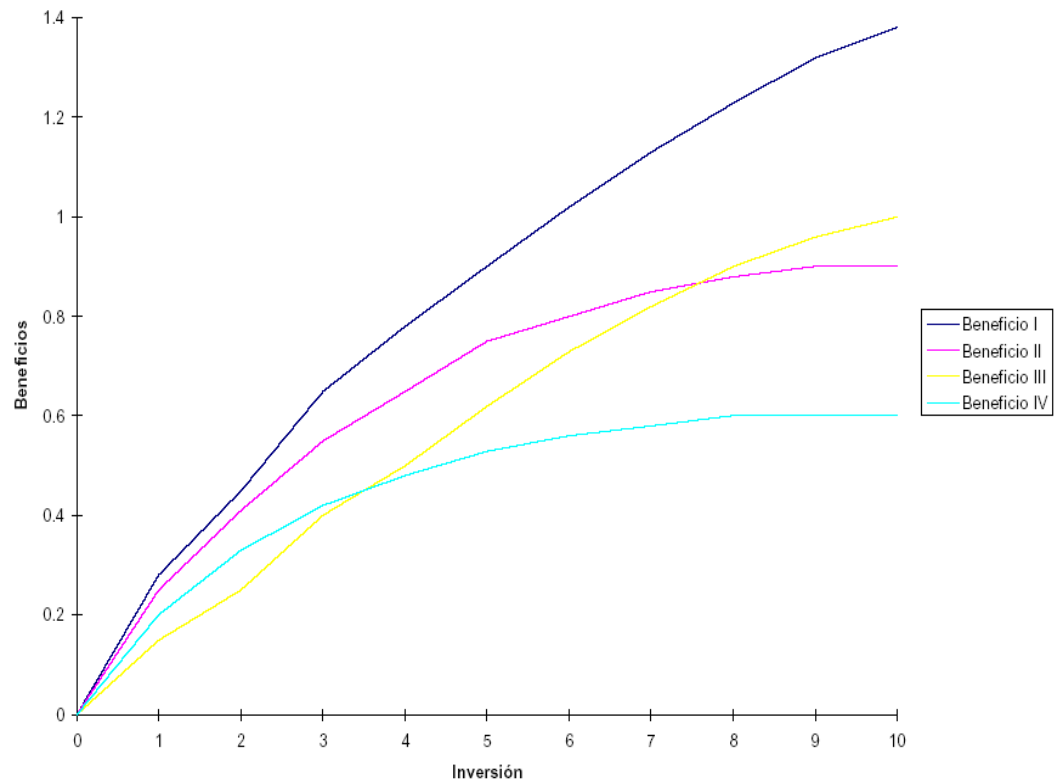


Figura 2.9. Gráfica de los valores de la Tabla

Aunque éste es sólo un sencillo ejemplo del uso del algoritmo genético para resolver problemas de optimización con restricciones, puede alcanzarse a percibirse el poder de la técnica en comparación con los métodos tradicionales de búsqueda.

Capítulo 3: Estado del Arte

En este capítulo se mostrarán las técnicas relacionadas al trabajo que se está realizando. Se hará referencia al conocimiento disponible hasta el momento en torno a nuestro tema, además de los resultados de investigaciones recientes de forma que integre y agregue claridad a nuestro trabajo.

3.1. Introducción

En este capítulo se dará una breve explicación a las técnicas más destacables relacionadas con los problemas de timetabling. Éstas fueron seleccionadas debido al gran éxito obtenido en la resolución de problemas de optimización combinatoria que es donde podemos considerar el nuestro. Como se mencionó antes, se trata de un problema clasificado como NP-Completo, por lo cual no existen algoritmos determinísticos que encuentren la solución óptima en un tiempo polinomial. Sin embargo, es posible utilizar estrategias con las cuales se pueden obtener soluciones aceptables. Las diferentes técnicas las podríamos dividir en dos grandes grupos:

- Técnicas tradicionales

Estos métodos que debido a su forma de búsqueda, se les denomina métodos completos, ya que recorren todo el espacio de búsqueda, es decir, encuentran todas las soluciones posibles a un determinado problema, sin embargo, éstas dependen del número de variables que influyen en el problema, es decir, su éxito depende directamente del número de variables involucradas.

Dentro de este grupo podemos encontrar los siguientes tipos: Programación entera, Programación Lineal, Backtracking, entre otras.

- Técnicas no tradicionales

Al contrario de las técnicas tradicionales, éstas no encuentran todas las soluciones posibles a un problema, es decir, acotan el espacio de búsqueda, por lo mismo caen en la categoría de métodos incompletos.

Dentro de este grupo podemos encontrar los siguientes tipos: Recocido Simulado (Simulated Annealing), Algoritmos Genéticos (Genetic Algorithms), Búsqueda Tabú (Tabu Search), Algoritmos Voraces (GRASP), Redes Neuronales (Neuronal Networks), Algoritmos Memético (Memetic Algorithms), entre otras.

3.2. Métodos Existentes para Resolver el Problema

En este caso revisaremos la Búsqueda Tabú, el Recocido Simulado y los Algoritmos Meméticos. Haremos una breve descripción de cada una de las técnicas escogidas y luego mostraremos ejemplos de su aplicación al problema de timetabling.

3.2.1 Búsqueda Tabú

La metaheurística Búsqueda Tabú es introducida y desarrollada por Fred Glover [28], la cual se utiliza mucho para la resolución de problemas de optimización cuya característica principal es la de salir del óptimo local.

"La filosofía de la Búsqueda Tabú es la de manejar y explotar una colección de principios para resolver problemas de manera inteligente. Uno de los elementos fundamentales de la Búsqueda Tabú es el uso de la memoria flexible. Desde el punto de vista de la Búsqueda Tabú, la memoria flexible

envuelve el proceso dual de crear y explotar estructuras para tomar ventaja mediante la combinación de actividades de adquisición, evaluación y mejoramiento de la información de manera histórica"... [7]

Este método está diseñado, como se mencionó, para salir del óptimo local, basado en el manejo y uso de una colección de principios que sirven para resolver el problema de manera inteligente. Esto se logra, haciendo uso de memoria flexible para involucrar dos procesos, el de la adquisición y el de mejoramiento de la información; así, al tener cierta historia de los caminos ya recorridos y de los óptimos encontrados, se puede evitar permanecer en los mismos espacios, y recorrer espacios nuevos para encontrar otras mejores soluciones.

El método de búsqueda tabú se basa en tres puntos principales:

- El uso de memorias diseñadas para permitir evaluar la información de búsqueda histórica.
- Un mecanismo de memoria que restringe y libera el proceso de búsqueda.
- La utilización de memorias de diferentes lapsos de tiempo: la de término corto, la de término intermedio y la de término largo, para guardar (por un tiempo) aquellas características que lograron una buena solución, y olvidando otras (ya transcurrido el tiempo de memoria) para permitir al método diversificarse dentro de nuevas regiones.

Además de esto, el método de búsqueda tabú tiene la habilidad de obtener resultados de alta calidad con equipo computacional modesto, lo que es una ventaja, ya que en muchas ocasiones no se cuenta con equipo computacional de alta calidad.

3.2.1.1. Descripción

El algoritmo Búsqueda Tabú requiere de una solución inicial, no es necesario que sea ni factible ni óptima. Esta solución dependerá del algoritmo que la genera, aunque es posible generar la solución inicial de manera aleatoria.

Una vez obtenida la solución inicial se procede a una etapa de mejora de ésta, moviéndose paso a paso hacia una solución que proporcione, en la medida de lo posible, el valor mínimo de la función objetivo (asumiendo que nos encontramos en un problema de minimización), para ejemplificar el método usaremos la función objetivo C .

Para explicar el procedimiento, representamos a cada solución por medio de un punto x (en algún espacio) y se define una vecindad $N(x)$ para cada punto x como un conjunto de soluciones adyacentes a la solución x .

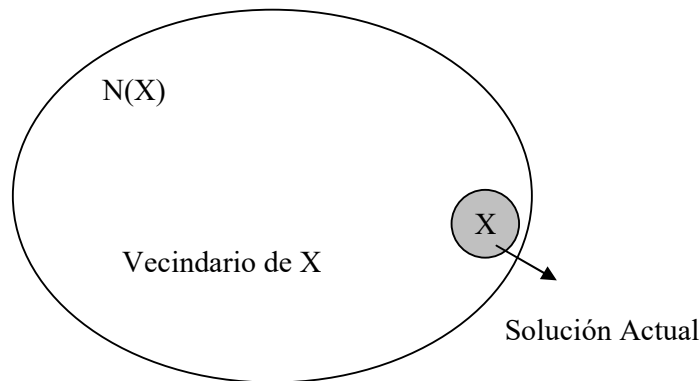


Figura 3.1. Vecindario de la solución actual

Es decir, el paso básico del procedimiento consiste en empezar desde un punto factible x , desde una solución inicial y generar un conjunto de vecinos, que son una serie de soluciones en $N(x)$ de esta vecindad, y así elegir la mejor x^* de ellas llamada “el mejor vecino” y se posiciona en este nuevo punto ya sea que $C(x^*)$ tenga o no mejor valor que $C(x)$. El hecho de permitir que en ocasiones la solución actual empeore es para evitar quedar atrapados en un óptimo local.

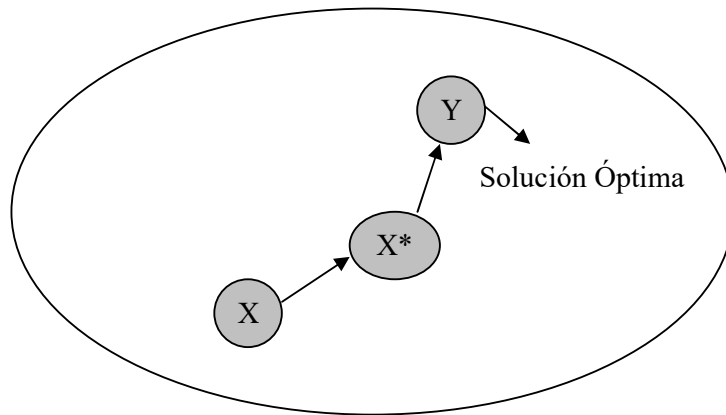


Figura 3.2. Trayectoria para encontrar la solución óptima

La característica importante y sobresaliente de la Búsqueda Tabú es precisamente la construcción de una lista tabú T de movimientos. Pero qué significa esto de Lista Tabú, pues se entiende por ella como todos aquellos movimientos que no son permitidos (movimientos tabú) en la presente iteración. De ahí su nombre "Tabú". La utilidad de esta lista es evitar los posibles movimientos que nos puedan regresar a algún punto de alguna iteración anterior, lo cual podría hacernos caer en un ciclo. El buen manejo de la lista tabú nos lleva a la exploración de nuevas regiones.

Ahora bien, un movimiento permanece como tabú sólo durante un cierto número de iteraciones, de forma que se tiene la Lista Tabú T .

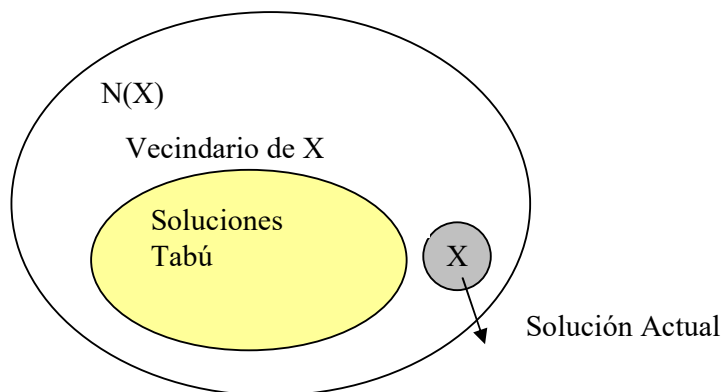


Figura 3.3 Lista Tabú

Es decir, la lista Tabú debe ser dinámica ya que pasado un determinado número de iteraciones, la búsqueda está en una región distinta y las soluciones antiguas pueden liberarse del status tabú.

También es importante seleccionar el tamaño de nuestra Lista Tabú, generalmente se toman valores dependiendo del tamaño del problema. De cualquier forma, constituye un parámetro importante cuya influencia habría que analizar y de la cual podría depender la evolución del algoritmo.

Dentro de esta Lista Tabú es importante, también, seleccionar qué se va a almacenar en ella, ya que, por ejemplo, almacenar la descripción completa de las últimas soluciones y comprobar alguno de estos movimientos que se encuentran en la lista puede ocupar mucho tiempo y memoria en nuestro sistema computacional. Como alternativa se puede almacenar sólo un atributo representativo del movimiento. Tanto los atributos que se consideren, como la forma de almacenarlos, dependerán del problema a resolver.

El criterio de aspiración consiste en dar un poco de flexibilidad a la Búsqueda Tabú. El estatus tabú de una solución puede ser ignorado si ciertas condiciones se cumplen, éstas se basan principalmente en admitir una solución, considerar una aspiración a pesar de estar clasificadas como Tabú. Esto es, que cualquier solución que sea mejor a cualquiera de las encontradas merece ser considerada, incluso si para alcanzarla debamos utilizar un movimiento prohibido.

En resumen, el método de Búsqueda Tabú debe tener una memoria adaptativa, que se caracteriza por su selectividad; su abstracción y descomposición; el tiempo, aquí es donde entra la diferenciación entre corto y largo plazo.

La serie de movimientos en que se rechazan, en específico la hecha en la Lista Tabú, se denomina de memoria a corto plazo, ya que esta búsqueda sólo se basa en una búsqueda local. Y descartando los movimientos encontrados dentro de nuestra lista tabú. Para problemas más complejos se pueden considerar estrategias de memorias a plazos más largos.

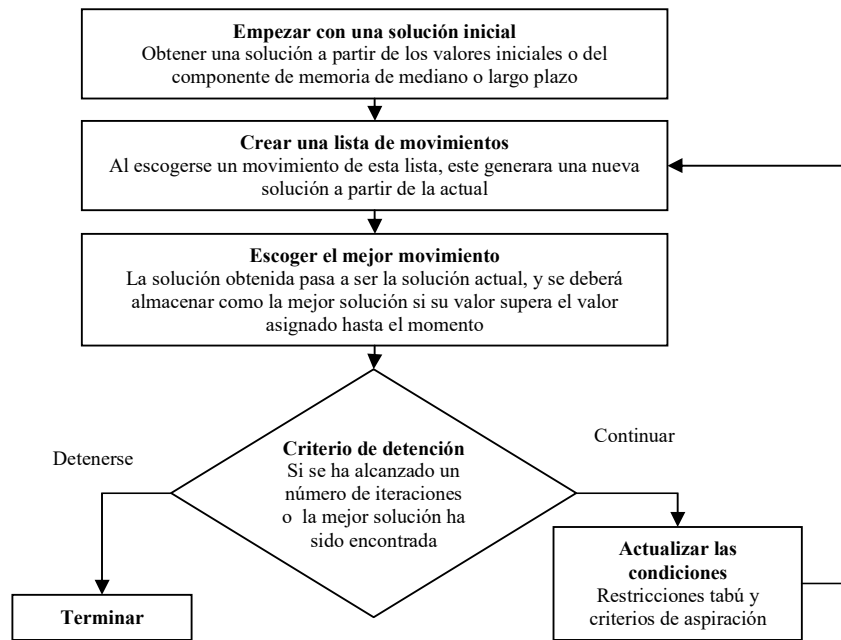


Figura 3.4 El componente de memoria de corto plazo de la búsqueda tabú

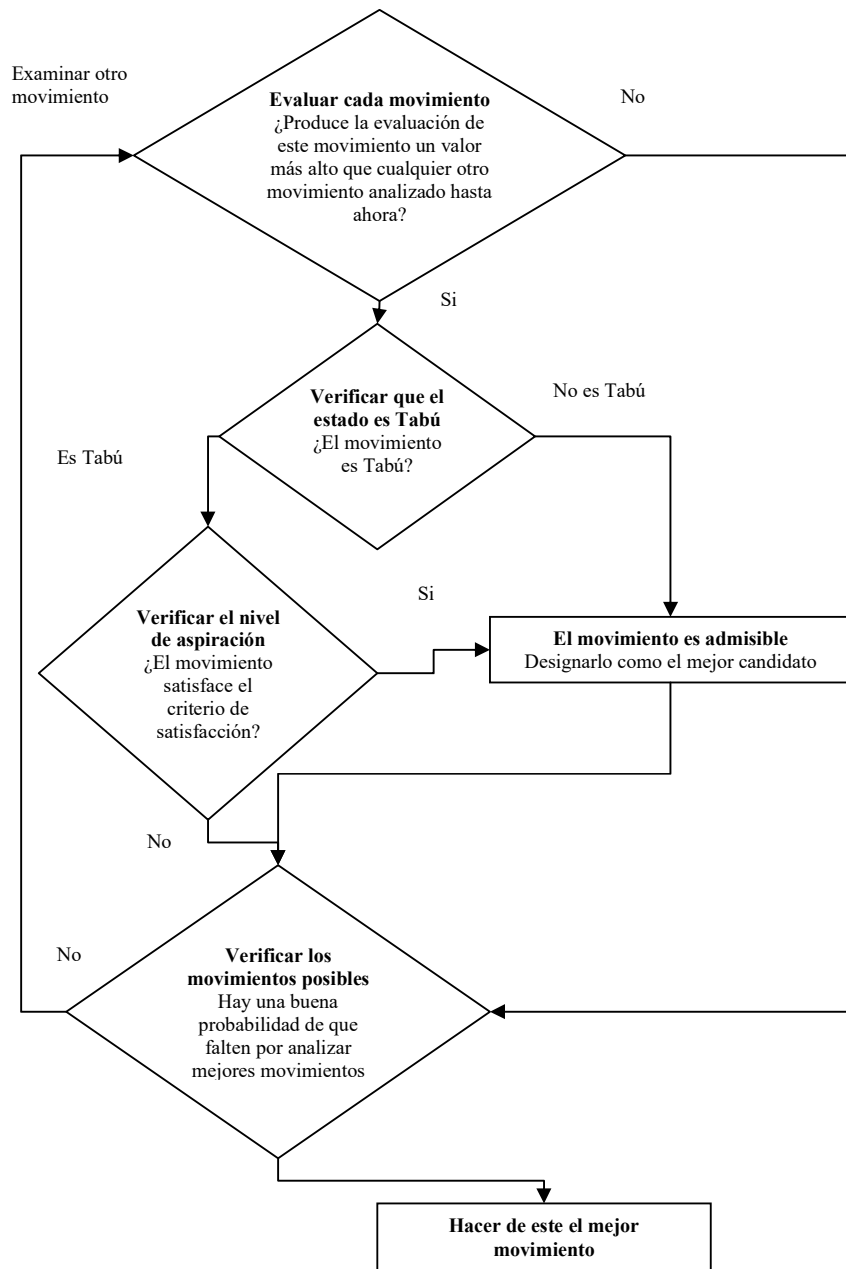


Figura 4.5. Selección del mejor movimiento posible

La memoria a largo plazo es aquella memoria basada en frecuencias, además que diversifica la búsqueda sobre regiones poco exploradas. Una forma común de hacer esta diversificación consiste en reiniciar cada determinado tiempo la búsqueda desde puntos elegidos aleatoriamente. Otro método, propone registrar los atributos de los movimientos más utilizados en los anteriores movimientos, penalizándolos a través de una lista tabú a largo plazo.

La heurística de Búsqueda Tabú requiere de una exploración sensible, que consiste en la imposición estratégica de limitaciones, un enfoque concentrado en buenas regiones y buenas características de las soluciones, caracterización y exploración de nuevas regiones prometedoras, patrones de búsqueda no monótonos e integración y extensión de soluciones.

Las estrategias de diversificación se basan en el cambio de las reglas, de modo que favorezcan combinaciones de movimiento, para así, de cierta forma evitar algún ciclo que se pueda dar. Las estrategias de diversificación están diseñadas para llevar la búsqueda hacia nuevas regiones. La mayoría de las veces son basadas en modificar los patrones de elección para así llevar a la solución de atributos que no se hayan tomado en cuenta.

Como criterio de terminación se puede tener en cuenta dos o más condiciones, como las que a continuación se anuncian:

- Que se haya realizado un número de iteraciones k sin mejorar el valor de la función objetivo C .
- Cuando se considere que el valor obtenido es cercano al óptimo esperado.
- Un número de iteraciones definido al inicio.

Algo que es básico e indispensable para implementar la Búsqueda Tabú es que el solucionador del problema, el que usará la meta heurística de búsqueda tabú, debe estar familiarizado con el problema, conocer su naturaleza y la forma de las soluciones factibles, de tal manera que pueda sugerir la configuración de la lista tabú, del vecindario y del criterio de aspiración.

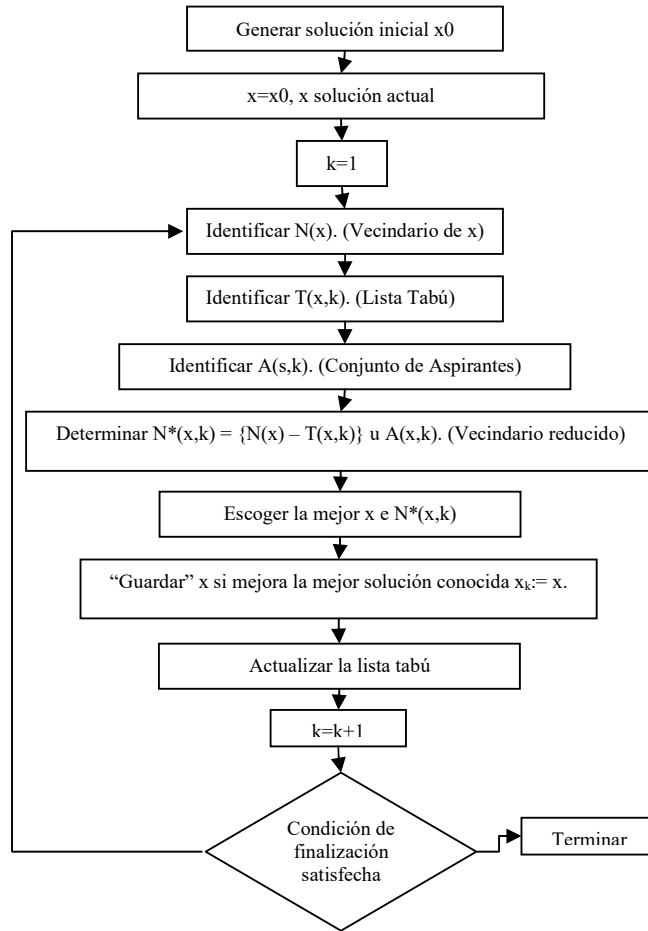


Figura 3.6. Algoritmo de la Búsqueda Tabú Simple

3.2.1.2. Aplicaciones de la búsqueda tabú al problema de la generación de horarios

La versatilidad de la Búsqueda Tabú la hace una técnica muy utilizada para resolver el problema del timetabling, como ejemplo tenemos el trabajo de Marcos Gil Tallavo y Amadís Antonio Martínez [6] donde toma en cuenta como recursos a combinar en el problema a los cursos, horarios y aulas.

Para representar a un horario de clases, definen un arreglo $H_{nd \times |U| \times ndh}$ de enteros, donde nd es el número de días, $|U|$ es el número de aulas y ndh es el número de horas por día. Cada valor h_{ijk} en el arreglo representa la sección, entendemos por esta a un grupo de estudiantes matriculados a un curso específico, que esta

asignada para un curso en el día i , en el aula j a la hora k . En el caso de que no se tenga ninguna sección asignada se tiene $h_{ijk} = 0$.

En la siguiente figura se presenta una ilustración de la estructura de datos. Además de esta estructura se definieron otras estructuras auxiliares, las cuales redundan cierta información pero permiten verificar las restricciones, tanto obligatorias como deseables, de una manera mas eficiente.

Día 1							...							Día nd						
horas														horas						
	1	2	3	4	...	nhd									1	2	3	4	...	nhd
aulas	1	s_2	s_2		...									1	s_2	s_2		...		
	2	s_1	s_1	s_1		s_8								2					...	
	3	s_3	s_3	s_5	s_5	...								3	s_1	s_1			...	s_7
	...	.	.	.	.	.								...	.	.	.	.	.	.
	U	s_7	s_7	s_7	...	s_9								U	s_4	s_4	s_5	s_5	...	

Figura 3.7. Estructura de un horario de clases

A continuación describiremos los componentes del algoritmo que fueron modificados para adaptarlo a su problemática:

Para la construcción de la solución inicial el algoritmo usa básicamente un procedimiento greedy (voraz), en el que se asignan primero las secciones con más restricciones, tomando días y aulas aleatorios y probando cada hora secuencialmente.

Para definir los movimientos se estableció el de intercambio de bloques. Este intercambio puede ser entre dos bloques de dos horas, dos bloques de tres horas, un bloque de tres horas y otro de dos horas con una hora libre antes o después del bloque, dos bloques de dos horas con una hora libre antes o después del bloque, o un bloque de dos o tres horas y un bloque libre del mismo tamaño. La vecindad de una solución viene dada por cierta cantidad de movimientos aleatorios. En las Figuras 3.8 y 3.9 se presentan ejemplos de intercambio de bloques.

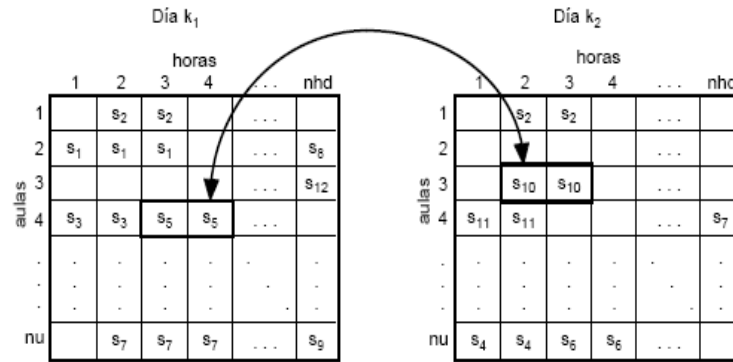


Figura 3.8. Ilustración del movimiento(a)

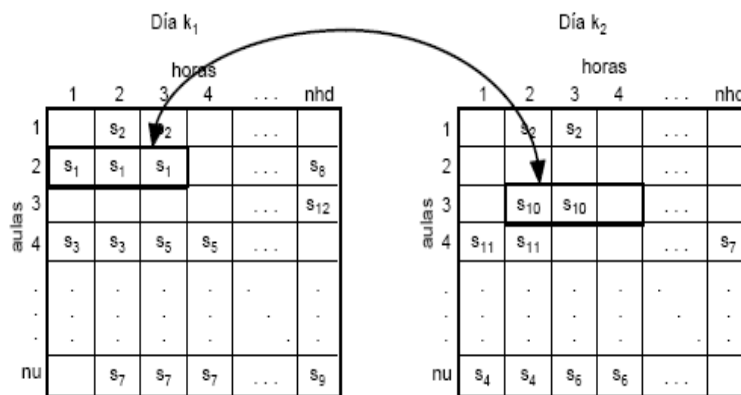


Figura 3.9. Ilustración del movimiento (b)

Como condición tabú única se tiene que cuando alguno de los bloques de movimiento haya sido recientemente utilizado, no se podrá usar después de un tiempo. De acuerdo con esto, después de intercambiar dos bloques, los mismos no podrían ser utilizados en un movimiento durante la cantidad de iteraciones dada.

Como único criterio de aspiración se tiene el criterio de la mejor solución. Este significa que no se toma en cuenta el estatus tabú de una solución cuando esta tiene menor costo que la mejor solución encontrada hasta el momento

Se definieron dos condiciones de terminación del ciclo principal del algoritmo:

- Se han realizado cierto número máximo de iteraciones.

- Si se llega a cierto número de iteraciones sin mejorar la mejor solución encontrada en toda la búsqueda, se regresa a esa mejor solución y se hace una intensificación de la búsqueda en la vecindad, por medio de la duplicación del número de movimientos aleatorios por iteración. Si al realizar esta intensificación aun no se encuentra una mejor solución, el ciclo se detiene.

Como conclusiones al trabajo se demostró que el algoritmo encontró mejores soluciones que las realizadas con el método manual en un tiempo razonable, además de mostrar los beneficios de combinar un procedimiento greedy (voraz) aleatorizado (utilizado para generar la solución inicial) con las estrategias de intensificación y diversificación y con estructuras de memoria a corto y largo plazo.

Tenemos el trabajo realizado por López Ceballo y Paulina Danae [15] en el cual se toma en cuenta a los docentes, cursos y el horario asignado. En este definen “Sea X_{ijk} una variable entera la cual toma el valor uno si se programa al docente i en el horario j y se le asigna el curso k y toma el valor cero en cualquier otro caso” como estructura que representara la solución.

Definen S como el conjunto de materias que se le programaron al docente i en los diferentes horarios que tenga disponibles, entonces tendríamos que:

$$S \leq \sum X_{irk} \forall j = 1, 2, \dots, n$$

Definen $C(S(x))$ como la función que genera la información de a qué maestros se le asigna qué materia y cuantas veces, por lo que:

Mínima carga académica del maestro $\leq C(S(x)) \leq$ Máxima carga académica del maestro.

Dado que la variable X_{ijk} toma el valor uno si se programa al docente i en el horario j y se le asigna el curso k , entonces tendríamos que:

$$\sum X_{rjk} = 0 \text{ ó } \sum X_{rjk} = 1$$

Y dado que S representa el número de cursos programados para el maestro i , en el horario j , tendríamos:

$$S \leq \sum X_{irj} \quad \forall j = 1, 2, \dots \text{ Máxima carga académica}$$

El cálculo de la función objetivo esta formado por varias componentes, las cuales se desean minimizar, estas son:

- El componente académico: se tiene un costo C_{ik} de asignar al docente i el curso k , en este caso mientras más deseable sea asignar a un maestro una materia, menor debe ser C_{ik} .
- El componente de horarios: una vez establecida una programación se puede medir lo compacto de ésta, midiendo el total de horas libres entre cursos para cada maestro.
- El componente de cursos distintos: se puede medir para cada maestro las materias distintas asignadas.

En cuanto a los movimientos posibles asignados en este trabajo son:

- Subir o bajar el valor de una variable (cero o uno): Para subir el valor de una variable se requiere que la suma de variables en ese grupo sea cero y todas las variables del grupo pueden subir.
- Subir una variable y bajar otra. Para bajar el valor de una variable se requiere que la suma del grupo sea uno y en ese grupo solo una variable pueda bajar.

Para los movimientos de las variables donde se utilizan dos variables, una que sube y otra baja se requiere que:

- Si ambas variables están en el mismo grupo, entonces la variable que baja deberá ser la que tiene asignado el número uno en el grupo.
- Si ambas variables están en grupos distintos se requiere entonces que la variable que baja esté localizada en el grupo cuya suma sea uno y la variable que sube puede localizarse en cualquier grupo cuya suma sea cero, esto asegura que la suma de un grupo sea cero y la suma en el otro sea uno.

El procedimiento se llevaría a cabo de la siguiente manera: Sobre todos los grupos se revisa la suma de las variables. Si ésta vale uno se busca en las variables del grupo, cual variable vale 1 y se prueba la disminución de su valor, haciendo la variable 0. Si la suma es cero se prueba el incremento de todas las variables (una por una) en el grupo, eligiendo por último la que genera "la mejor solución". Cada uno de los movimientos realizados es almacenado en la lista $T(x)$ tabú de tal manera que el movimiento anterior no podrá realizarse otra vez a menos que cumpla con el criterio de aspiración (es decir que genera la mejor solución encontrada en el grupo).

Para los movimientos de dos variables se procede en forma similar: Se comparan las sumas de dos grupos: si sus sumas son distintas, se busca en el grupo cuya suma sea uno la variable que vale uno, para disminuir su valor a cero. Y de forma similar a la anterior, se prueba aumentando el valor de cada una de las variables del otro grupo. Si el grupo es el mismo y la suma es cero, no se hace nada. Si la suma es 1 se procede en forma similar. Cada vez que se genere un movimiento la lista tabú se ira incrementando.

La lista tabú se implementa guardando dos valores, el primero indica las variables que suben su valor y el segundo las que bajan, cuando el movimiento es de una sola variable se puede representar como (0,1) si su valor baja ó (1,0) si su valor sube.

A continuación presentamos el pseudo código para entender mejor la solución planteada:

Introducir Datos $\leftarrow$ Base de datos inicial

Inicio

Paso 1. Formar los grupos de cada docente

Paso 2. Seleccionar un X_{ijk} en el primer grupo

Si $X_{ijk} = 0$ entonces sube, de otra forma baja

Paso 3. Guardar el movimiento en la lista tabú.

Paso 4. Verificar si $\sum X_{rjk} = 0$ ó $\sum X_{rjk} = 1$

Si $\sum X_{rjk} = 1$ continuar con el siguiente grupo

Si $\sum X_{rjk} = 0$ regresar al paso 2

Paso 5. Verificar si los cursos asignados al maestro i se encuentran en el rango

$$L1 \min(i) \leq \sum_{i,k} X_{rjk} \leq L1 \max(i)$$

Si se cumple, continuar con el siguiente paso

Si no, regresar al Paso 2

Paso 6. Medir la compactación de horarios del docente i en el horario j $\sum X_{rjk}$ para los j 's donde $X_{rjk} = 0$. Si X_{rjk} es mínima:

- Catalogar este arreglo como optimo
- Hacer los demás movimientos tabú
- Continuar con el siguiente paso

Paso 7. Medir el costo académico de los óptimos del paso anterior

$$\sum_{i,j,k} C_{ijk} X_{rjk}$$

Hacer óptimo el arreglo que tenga el menor costo.

Paso 8. Verificar si el número de aulas disponibles en el horario j anda en el rango

$$L2 \min(j) \leq \sum_{j,k} X_{rjk} \leq L2 \max(j)$$

Si no se cumple regresar al Paso 2

Si se cumple continuar con el siguiente paso

Paso 9. Verificar si en el horario j se programaron $C2(j)$ cursos de la materia k

$$\sum_i X_{rjk} \leq C2(j)$$

Si no se cumple regresar al Paso 2

Si se cumple continuar con el siguiente paso

Paso 10. Verificar si para la materia k se programaron los cursos $C3(j)$ en la mañana y $C4(j)$ en la tarde, tomando en cuenta que L es el último horario de la mañana.

$$\sum_i X_{rjk} \leq C3(j)$$

$$\sum_i X_{rjk} \leq C4(j)$$

Si no se cumple regresar al paso 2.

Si se cumple catalogar este arreglo como la mejor solución

Fin.

En este trabajo se concluyó que El método de búsqueda tabú mostró tener siguiente las ventajas se pueden apreciar en:

- La rapidez en la captura de datos,
- Permite manipular un número grande de requerimientos y restricciones en la programación que otros programas no pueden,
- Permite modificar la programación cuantas veces se quiera ya que,
- La obtención de resultados no supera los dos segundos y
- Los resultados obtenidos fueron mucho mejores que los que se habían obtenido en semestres anteriores, ya que se compactaron muchos horarios

3.2.2 Algoritmos Meméticos

En 1989, Moscato [19] presentó el término Algoritmo Memético para describir a la combinación de Algoritmos Genéticos con métodos de optimización local. De esta manera, los Algoritmo Memético son

considerados como una extensión de Algoritmos Genéticos que aplica procesos separados para refinar a los individuos.

Los Algoritmo Memético fueron inspirados por modelos de adaptación en sistemas naturales que combinan la adaptación biológica o genética de una población con el aprendizaje que los miembros de esta población pueden lograr durante su tiempo de vida. La elección del nombre de estos algoritmos se basa en el concepto de meme definido por Richard Dawkin [1]. Un meme se refiere a una unidad de evolución cultural que puede lograr refinamiento local en los individuos de una población dada [1]. Una diferencia clave que existe entre genes y memes consiste en que antes de que un meme sea transmitido, este último es típicamente adaptado por el individuo que lo transmite en base a la forma en que este individuo piensa, entiende y procesa al meme. Sin embargo, los genes son transmitidos sin que el individuo actúe sobre ellos.

En el contexto de problemas de optimización, un meme representa a una estrategia de aprendizaje u optimización local que puede ser empleada para mejorar a un individuo. De esta forma, un modelo de adaptación memética logra aprovechar la plasticidad que poseen los individuos. Esto último es algo que un modelo genético no puede capturar.

La integración de un Algoritmos genéticos con métodos de optimización local ha mostrado que permite acelerar el descubrimiento de buenas soluciones, o alcanzar soluciones que serían inalcanzables si sólo se aplicara evolución genética [8]. Esto se debe a que el refinamiento local u optimización de los individuos permite que el Algoritmos Meméticos desarrolle una exploración mucho más dirigida hacia las regiones del espacio de búsqueda que poseen las mejores soluciones. Desde un punto de vista de optimización, los Algoritmos Meméticos han mostrado ser más eficientes, requieren menos generaciones para encontrar un óptimo, y más efectivos, identifican soluciones de mayor calidad, que los tradicionales Algoritmos Genéticos sobre varios dominios de aplicación. [17]

Como se mencionó anteriormente, puede ser muy beneficioso combinar la búsqueda desarrollada por un Algoritmo Genético con métodos que permitan refinar a cada solución de manera individual. Sin embargo, los beneficios logrados deben ser balanceados con el incremento en el nivel de

complejidad del algoritmo. De esta manera, es necesario considerar con mucho cuidado cómo se desarrollará exactamente la integración mencionada.

En la siguiente figura se muestra la estructura genérica de un Algoritmo Memético, la figura mencionada permite identificar la estructura básica de un Algoritmo Genético. En esta estructura, se han identificado y marcado con círculos los lugares donde la integración puede tomar lugar. Cada uno de los lugares marcados provee una oportunidad de integrar métodos de optimización. Por ejemplo, la población inicial podría contener soluciones provenientes de heurísticas diseñadas especialmente para el problema que se intenta resolver, y los operadores de recombinación y mutación podrían ser realizados con restricciones específicas del dominio. Además, los métodos de optimización local podrían ser aplicados a uno o a todos los conjuntos intermedios de soluciones. Por ejemplo, podrían ser aplicados sobre el conjunto de soluciones obtenido luego de la recombinación, sobre el conjunto de soluciones obtenido luego de la mutación, o sobre ambos conjuntos. Sin embargo, la forma más aceptada de integración consiste en aplicar una o más fases de optimización local, basada sobre algún parámetro de probabilidad, a miembros individuales de la población en cada generación.

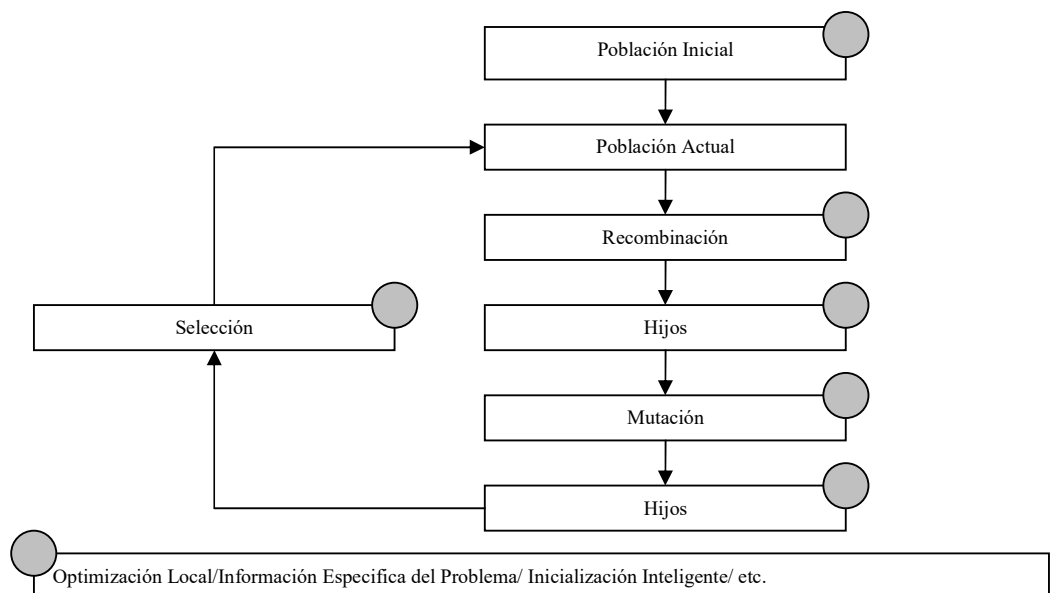


Figura 3.10. Estructura de un Algoritmo Memético

3.2.2.1. Métodos de Optimización Local

En el contexto de un Algoritmo Memético, cada individuo de una población contiene una carga genética heredada y, además, está ligado a un meme o a más de un meme. En otras palabras, cada individuo representa a una solución posible y, en forma adicional, posee algún método que le permitirá optimizar o refinar esa solución. De esta manera, el concepto de meme se refiere a un método de optimización individual desarrollado de manera local.

Es posible considerar distintos métodos de optimización como, por ejemplo, métodos exactos, métodos de aproximación, métodos de perturbación, etc. [13]. Sin embargo, la mayoría de los Algoritmos Meméticos hallados en la literatura utilizan heurísticas de búsqueda local como métodos de optimización. Tal es la aceptación de estos métodos de búsqueda local que muchos autores definen a los Algoritmos Meméticos como un Algoritmo Genéticos que incluye una o más fases de búsqueda local dentro de su ciclo evolutivo [13].

Al utilizar heurísticas de búsqueda local, se asume que la búsqueda evolutiva provee una amplia exploración del espacio de búsqueda mientras que la búsqueda local puede desarrollar una especie de explotación sobre la base de soluciones prometedoras. Estos métodos de búsqueda local permiten refinar una solución en función de la vecindad que esta última posee. En otras palabras, dada una solución particular s_0 un método de búsqueda local explora la vecindad de s con el objetivo de hallar una solución s' mucho mejor que s . Luego, se deberá establecer de que manera s' reemplaza a s .

Este reemplazo puede ser llevado a cabo siguiendo el enfoque de Baldwin o el de Lamarck. Considérese que el individuo i pertenece a la población P en la generación t y que el fitness de i es $f(i)$. Adicionalmente, supóngase que la búsqueda local sobre i produce un nuevo individuo i' con $f(i') < f(i)$ para un problema de minimización. En este momento, será necesario elegir de que forma

el individuo i' reemplazará a i . Más precisamente, será necesario elegir entre dos alternativas posibles.

La primera alternativa consiste en reemplazar al individuo i con el individuo i' , en este caso la información genética de i es perdida y reemplazada con la información de i' . La segunda alternativa consiste en mantener la información genética de i y cambiar su valor de fitness por el valor de fitness de i' , de esta manera $f(i') = f(i)$.

La primera alternativa presentada es conocida como aprendizaje de Lamarck, mientras que la segunda alternativa es conocida como de Baldwin. Según las pruebas realizadas, para algunos casos, el enfoque de Baldwin es más eficiente para converger en un óptimo local pero más lento que el enfoque el de Lamarck.

3.2.2.2. Anatomía de un Algoritmo Meméticos

Los Algoritmos Meméticos combinan conceptos de diferentes meta heurísticas tal como se menciona. En particular, la relación con los Algoritmos Genéticos resulta especialmente significativa, es por este motivo comenzaremos la descripción de estas técnicas abordando los aspectos heredados de este.

Un Algoritmo Memético mantiene en todo momento una población de diversas soluciones al problema considerado. Llamaremos agente a cada una de las mismas. Esta denominación es una extensión del termino individuo tan comúnmente empleado en el contexto de los Algoritmos Genéticos, y permite capturar elementos distintivos de los Algoritmo Memético que se mostrarían mas adelante (por ejemplo, un agente puede contener mas de una solución al problema considerado [25]). Estos agentes se interrelacionan entre si en un marco de competición y de cooperación, de manera muy semejante a lo que ocurre en la naturaleza entre los individuos de una misma especie.

Cuando consideramos la población de agentes en su conjunto, esta interacción puede ser estructurada en una sucesión de grandes

pasos temporales denominados generaciones. La Figura 3.10 muestra el esquema de una de estas generaciones.

```
Proceso Paso_Generacional(  
    poblacion Agente[],  
    operadores Operador[]  
)  
    criadores, nueva_poblacion Agente[];  
  
    criadores ← Seleccion(poblacion);  
    nueva_poblacion ← Reproducir(criadores, operadores);  
    poblacion ← Actualizar(poblacion, nueva_poblacion);  
FinProceso
```

Figura 3.11. Esquema de una generación

Tal como se muestra, cada generación consiste en la actualización de la población de agentes, usando para tal fin una nueva población obtenida mediante la recombinación de las características de algunos agentes seleccionados. Precisamente, este componente de selección es, junto con el paso final de actualización, el responsable de forzar la competición entre agentes.

Más concretamente, la selección se encarga de elegir una muestra de los mejores agentes contenidos en la población actual. Esto se realiza mediante el empleo de una función guía F encargada de cuantificar cuan bueno es cada uno de los agentes en la resolución del problema abordado.

```
Funcion Reproducir(  
    poblacion Agente[],  
    operadores Operador[]  
) Agente[]  
    buffer Agente[];  
    j Entero;  
    Para j ← 1 Hasta longitud(operadores) Hacer  
        buffer[j] ← Aplicar(operadores[j], buffer[j-1]);  
    FinPara  
    Retornar buffer[longitud(operadores)]  
FinFuncion
```

Figura 3.12. Generación de la nueva población

Por su parte, el reemplazo o actualización incide en el aspecto competitivo, encargándose de la importante tarea de limitar el tamaño de la población, esto es, eliminar algunos agentes para permitir la entrada de otros nuevos y así enfocar la tarea de búsqueda. En este proceso también puede emplearse la información proporcionada por la función guía para seleccionar los agentes que se eliminaran.

Tanto la selección como el reemplazo son procesos puramente competitivos en los que únicamente varía la distribución de agentes existentes, esto es, no se crean nuevos agentes. Esto es responsabilidad de la fase de reproducción. Dicha reproducción tiene lugar mediante la aplicación de cierto número de operadores reproductivos tal como se muestra en la Figura 3.12. Como puede apreciarse, es posible emplear un número variado de operadores. No obstante, lo más típico es emplear únicamente dos operadores: recombinación y mutación. El primero es el responsable de llevar a cabo los procesos de cooperación entre agentes (usualmente dos, aunque es posible considerar grupos mas grandes). Dicha cooperación tiene lugar mediante la construcción de nuevos agentes empleando información extraída del grupo de agentes recombinados, y quizás alguna información externa. Precisamente, la inclusión de información externa no contenida en ninguno de los agentes involucrados es responsabilidad del operador de mutación. Básicamente, este tipo de operador generara un nuevo agente mediante la modificación parcial de un agente existente.

```
Proceso Optimizador_Local(  
    actual Agente,  
    operador Operador  
)  
    nuevo Agente;  
    Repetir  
        nuevo ← Aplicar(operador,actual);  
        Si Fg(nuevo)<Fg(actual) Entonces  
            actual ← nuevo;  
        FinSi  
    Hasta Que Terminacion_Local();  
FinProceso
```

Figura 3.13. Esquema de un optimizador local

En relación con los operadores de mutación, es posible definir una meta operador basado en la aplicación iterativa de un operador de mutación arbitrario sobre un agente. El empleo de estos meta operadores es uno de los rasgos mas distintivos de los Algoritmos Memético. Concretamente, dichos meta operadores iteran la aplicación del operador de mutación, conservando los cambios que llevan a una mejora en la bondad del agente, motivo por el cual son denominados optimizadores locales. La Figura 3.13 ilustra el proceso. Como puede verse, la iteración en la aplicación del operador de mutación y la subsiguiente conservación de los cambios favorables se realiza hasta que se alcanza un cierto criterio de terminación (un número de iteraciones fijo, un cierto número de iteraciones sin mejora, haber alcanzado una mejora suficiente, etc.).

Estos optimizadores locales pueden considerarse como un operador más, y como tales pueden emplearse en diferentes fases de la reproducción. Por ejemplo, pueden usarse tras la aplicación de otros operadores simples de recombinación y mutación, aplicarse solo a un sub. conjunto de los agentes o únicamente ser aplicado al final del ciclo reproductivo. En cualquier caso, es la existencia de estos optimizadores individuales lo que justifica el empleo de la denominación agente en la jerga de los Algoritmos Meméticos:

El algoritmo puede caracterizarse como una colección de agentes que realizan exploraciones autónomas del espacio de búsqueda, cooperando ocasionalmente a través de la recombinación, y compitiendo continuamente por los recursos computacionales a través de la selección y el reemplazo.

Tras haber descrito el paso generacional básico, podemos ahora considerar una visión global de un Algoritmo Memético basada en la iteración de este procedimiento, tal como se muestra en la Figura 3.14.

```
Funcion MA(  
    tamaño Entero,  
    operadores Operador[]  
    ) Agente  
    poblacion Agente[];  
  
    poblacion ← Iniciar_Poblacion(tamaño);  
    Repetir  
        poblacion ← Paso_Generacional(poblacion ,operadores);  
        Si Convergencia(poblacion) Entonces  
            poblacion ← Reiniciar_Poblacion(poblacion);  
        FinSi  
    Hasta Que Terminacion_MA();  
  
    Retornar i_esimo_Mejor(población,1)  
FinFuncion
```

Figura 3.14. El esqueleto genérico de un MA

La generación de la población inicial puede acometerse de diferentes formas. Por ejemplo, pueden crearse longitud(pob) agentes al azar, o emplear las soluciones proporcionadas por heurísticas existentes. Una posibilidad mas sofisticada es el empleo de optimizadores locales para tal fin, tal como se ilustra en la Figura 3.14:

```

Funcion Iniciar_Poblacion(
    tamaño Entero
) Agente[]
población Agente[];
agente Agente;
j Entero;
Para j ← 1 Hasta tamaño Hacer
    agente ← Agente_Aleatorio()
    población[j] ← Optimizador_Local(agente);
FinPara
Retornar población

```

Figura 3.15 Generación guiada de la población inicial

La función para la reiniciación de la población es otro de los componentes fundamentales de los Algoritmos Meméticos, ya que de el dependerá que se haga un uso apropiado de los recursos computacionales del sistema, o por el contrario estos se malgasten en la explotación de una población de agentes que hayan alcanzado un estado degenerado, por ejemplo, con una gran similitud entre todos los agentes de la población. Esto se lo que se conoce como convergencia del Algoritmo Memético. Una vez se ha detectado la convergencia del Algoritmo Memético, la población de agentes se reinicia, conservando una porción de la misma, y generando nuevos agentes para completarla (Figura 3.16).

```

Funcion Reiniciar_Poblacion(
    población Agente[]
) Agente[]
nueva_población Agente[];
j,cons Entero;
cons ← Tamaño_Consevar(longitud(población))
Para j ← 1 Hasta cons Hacer
    nueva_población[j] ← i_esimo_Mejor(población,j)
FinPara
Para j ← cons + 1 Hasta longitud(población) Hacer
    nueva_población[j] ← Agente_Aleatorio();
    nueva_población[j] ←
        Optimizador_Local(nueva_población[j]);
FinPara
Retornar nueva_población

```

Figura 3.16. Reiniciación de la población en un Algoritmo Memético

3.2.2.3. Aplicaciones de los Algoritmos Meméticos al problema de la generación de horarios

Los Algoritmos Meméticos se abren como una nueva opción para resolver los problemas del timetabling, como ejemplo tenemos el trabajo de Alpay Alkan y Ender Özcan [24] los cuales abordan el problema de asignación de horarios, cursos y aulas. Estos consideran el problema como una 3-tupla (V, D, C) donde $V = \{v_1, v_2, \dots, v_n\}$ un número finito de cursos a dictar en un departamento, facultad o universidad. $D = \{d_1, d_2, \dots, d_i, \dots, d_n\}$, es un grupo finito de conjuntos de variables, como $G = \{t_1, t_2, \dots, t_m\}$, por ejemplo, que representa el horario de inicio de los cursos a dictar, entonces un posible dominio de para cada variable puede ser $d_i \subseteq G$ y C es el conjunto de restricciones a satisfacer. Por ejemplo $d_i \subseteq G \times S$ puede ser el dominio de una variable, donde S representa el grupo de aulas disponibles. Por ello definen su problema de timetabling como la búsqueda de la mejor asignación (v_i, t_j) para cada $v_i \in V$, tal que todas las restricciones son satisfechas. Esta asignación implica que el dictado de un curso v_i inicia en un tiempo t_i .

Definen 6 grupos diferentes grupos de restricciones:

- De Exclusión, las cuales representan la exclusión de algunas posibles soluciones, como por ejemplo que no se dicte estructura de datos los días martes.
- Predefinidas, representan las asignaciones predefinidas de ciertas variables, como que el curso de electrónica se dicte los viernes de las 14 a las 17.
- Límites, son las restricciones más usadas en los problemas del timetabling y tenemos por ejemplo que dos cursos no pueden iniciar a la misma hora en la misma aula.
- De orden, estas representan las restricciones de orden entre el dictado de cursos. Por ejemplo el curso de teoría de algoritmos

debe de programarse a la misma hora que el curso de Base de Datos.

- De Distribución de eventos, estas restricciones nos dicen como deben de distribuirse las horas en la semana
- De Atributos. Estos representan las restricciones aplicadas entre atributos del curso a dictar y / o los atributos de su asignación. Por ejemplo asumamos que un atributo del curso sea el total de alumnos que pueden tomar el curso y un atributo del aulas sea su capacidad entonces una posible restricción es que el total de alumnos que se puedan a escribir al curso sea menor que la capacidad del aula.

La representación usada consiste en un conjunto de las clases a dictar, las cuales están agrupadas, en los cursos respectivos, secciones y departamentos de manera jerárquica. Cada gen denota el tiempo de inicio del dictado del curso en un aula. Un curso puede ser dictado más de una vez en la semana.

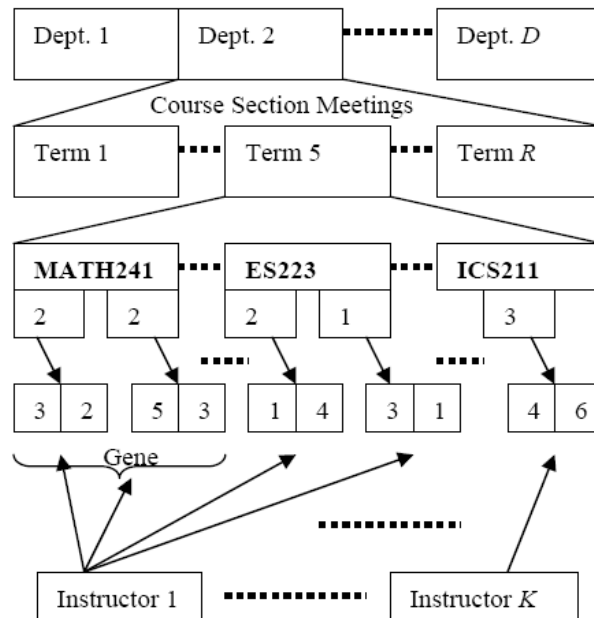


Figura 3.17. Representación de la solución

Un óptimo timetabling es aquel que satisfaga todas las restricciones. Sea p_j que represente la penalidad asociada con la restricción $i \in C$ perteneciente al tipo de restricción j y $g_i(T)$ representa el

número de violaciones en la solución T debido a la restricción i y w_j representa el peso aplicado a p_j :

$$f(T) = \sum_{\forall j \forall i} w_j p_j g_i(T)$$

Para la generación de la población inicial se usa ciertas restricciones para sesgar la solución aleatoria a una solución factible. Después de esta inicialización aleatoria la población es pasada através de un método de ascenso a la colina. Un método adicional es implementado para proveer orientación a la búsqueda ajustando los valores de las penalidades w_j según la población inicial generada.

$$w_j = \forall y \in Poblacion \left\{ \frac{\sum g_i(y)}{\sum f(y)} \right\}$$

Utilizan operadores convencionales de reproducción y mutación pero posterior a cada uno de estos utilizan un procedimiento de ascenso a la colina para mejorar el individuo hasta un óptimo local. La idea es crear un método de ascenso a la colina por cada tipo de restricción y combinarlo bajo una solo. El modo de aplicar el método de ascenso a la colina es el siguiente, seleccionar el método según la restricción que tenga un mayor numero de violaciones, luego seguimos los siguientes pasos:

- Aplicar el método para eliminar todas las violaciones del tipo de restricción seleccionado produciendo un nuevo individuo.
- Si el intento anterior no produce mejoras ignoramos el nuevo individuo. Según la restricción, seleccionar el bloque de genes que podría estar causando mas violaciones y tratar de corregirlo remplazándolo.
- Si lo anterior no funciona escoger un único gen del bloque de genes que podría estar causando mas violaciones y tratar de corregirlo remplazándolo.

Si la aptitud de un individuo mejora en cualquier caso, se vuelve a aplicar el método de ascenso a la colina en este individuo. En cuanto a la estrategia de selección utiliza el operador de elitismo.

Como resultados posteriores a la prueba del algoritmo con información real se logro que el tiempo computacional se reduzca y la calidad de la solución incremente comparado contra un algoritmo evolutivo sin optimizadores locales.

3.2.3 Recocido simulado

Este tipo de algoritmos aparece a principios de los años 80 [29]. Se trata de un tipo de modelo de resolución para la optimización de problemas de tipo combinatorio con mínimos locales. Su aproximación consiste en generar aleatoriamente una solución cercana a la solución actual (o en el entorno de la solución) y la acepta como buena si consigue reducir una determinada función de costo, o con una determinada probabilidad de aceptación. Esta probabilidad de aceptación se irá reduciendo con el número de iteraciones y está relacionada también con el grado de empeoramiento del coste.

“El recocido simulado de un objeto para mejorar alguna de sus propiedades consiste en producir al azar cambios en su descripción. Si es posible definir una temperatura en relación a tal propiedad, el proceso comienza a altas temperaturas, en las que todos los cambios aleatorios son aceptados (y en consecuencia el aspecto del objeto inicial cambia continuamente); y continúa con un paulatino y lento descenso de la temperatura, de manera que cada vez el número de cambios aceptados disminuye más y más y una forma particular del objeto acaba por emerger. Finalmente, cuando la temperatura es tan baja que ningún cambio se acepta, el algoritmo se detiene con el objeto inicial profundamente alterado, de hecho con el objeto que probablemente pueda encontrarse para la propiedad de interés. Esta situación inicial es el cero absoluto del objeto respecto a la propiedad”. [21]

Su fundamentación se basa en el trabajo de Nicholas Constantine Metrópolis en el campo de la termodinámica estadística. Básicamente, Metrópolis modeló el proceso de recocido simulando los cambios energéticos en un sistema de partículas conforme decrece la temperatura, hasta que converge a un estado estable (congelado).

Estos algoritmos se llaman de esta forma por su parecido en funcionamiento al proceso de enfriamiento metalúrgico: cuando se enfría un metal fundido

suficientemente despacio, tiende a solidificarse en una estructura de mínima energía (equilibrio térmico); según va disminuyendo la temperatura, las moléculas del metal tienen menor probabilidad de moverse de su nivel energético. La probabilidad de movimiento de una molécula de su estado viene dada por la función de Boltzmann $P(\Delta E) = e^{\Delta E / kT}$, que depende de la temperatura, de la diferencia de energía y de una constante K (la constante de Boltzmann).

Los algoritmos de enfriamiento lento simulado tienen algunas ventajas con respecto a otras técnicas de optimización global. De acuerdo con Elmohamed [20] entre las ventajas de estos algoritmos se pueden citar:

- Lo relativamente sencillo que resulta implementar este tipo de problemas.
- Su aplicabilidad a la mayoría de los problemas de optimización con una estructura combinatoria.
- Su capacidad para ofrecer soluciones razonablemente buenas a la mayoría de los problemas, aunque hay una cierta dependencia de la planificación del enfriamiento y los movimientos que se realicen.
- La facilidad con la que se puede combinar este tipo de algoritmos con otras técnicas heurísticas como los sistemas expertos, los algoritmos genéticos, las redes neuronales, etc., consiguiendo sistemas híbridos que pueden resultar de gran potencia en la resolución de problemas muy complejos.

Por otra parte, también se pueden citar algunos aspectos que pueden limitar su utilización:

- Se necesita elegir con mucho cuidado los movimientos que se realizan, así como los parámetros que se van a utilizar para tratarlo, como por ejemplo la tasa de enfriamiento.
- Una ejecución del problema puede requerir mucho tiempo de cálculo.
- Puede que sea necesario realizar muchas ejecuciones para encontrar una solución satisfactoria.
- Dependiendo de los parámetros elegidos, las soluciones que se van encontrando pueden ser poco estables, “saltando” mucho de unas a

otras sin encontrar una solución buena con la rapidez suficiente lo que obliga a retocar los parámetros con las distintas ejecuciones.

3.2.3.1. Estructura del algoritmo

La estructura básica de un algoritmo de enfriamiento lento simulado se puede escribir de acuerdo con el siguiente diagrama de flujo:

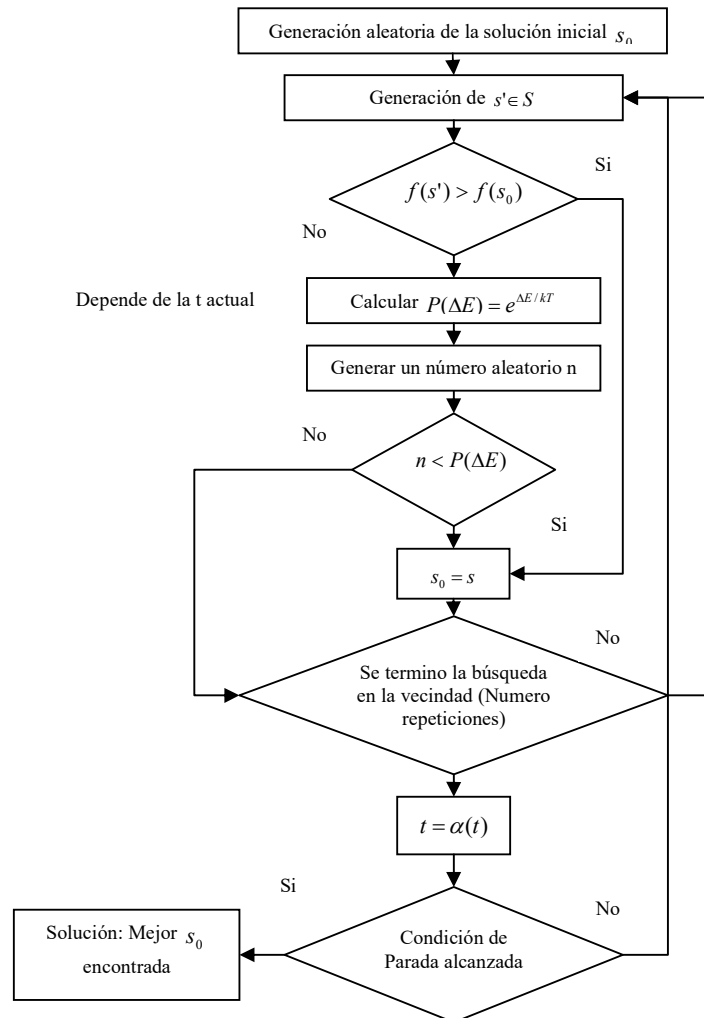


Figura 3.18. Diagrama de Flujo del Algoritmo de Recocido Simulado

En este diagrama se pueden diferenciar dos tipos de elementos que permiten afinar el algoritmo para que la ejecución del mismo sea la más eficaz de acuerdo con el problema que se desea resolver:

Elementos genéricos. Son elementos que no tienen una dependencia directa del problema, aunque hay que afinarlos para el problema concreto que se desea resolver.

- Temperatura inicial: t_0
- Proceso de enfriamiento: $\alpha(t)$
- Número de repeticiones.
- Condición de parada.

Elementos dependientes de problema. Son los elementos que definen de forma directa el problema que se está resolviendo. Definen un modelo del problema y la estructura del espacio de soluciones para el mismo.

- Espacio de soluciones: S
- Estructura de vecindad: S_i
- Función de coste: $f(s)$
- Solución inicial: s_0

La **temperatura inicial** define el momento en el que se inicia el proceso de enfriamiento. Esta temperatura inicial debe ser suficientemente alta para permitir el libre cambio de soluciones vecinas y, sobre todo, que la solución final sea independiente de la solución inicial.

Una regla práctica que se le suele poner a la temperatura inicial es que debe tener un valor tal que la proporción inicial de vecinos aceptados (tanto de mejora como de no mejora) tenga un alto valor (90% - 95%). Es decir, inicialmente todos los vecinos se aceptan con una probabilidad cercana a 1.

Lógicamente hay que elegir esta temperatura con cuidado porque si se elige una temperatura inicial demasiado alta se perderá tiempo realizando muchos movimientos que se aceptan siempre, lo que no resulta de mucho interés pues no se está persiguiendo ningún valor concreto.

Por otra parte, si se elige una temperatura inicial demasiado baja el proceso se mueve poco desde la solución inicial por lo que se puede quedar atrapado desde un inicio en una región desde la que sólo se pueda llegar a un óptimo local, por lo que se perderá la posibilidad de explorar otras regiones donde se encuentre alguno de los óptimos globales del sistema.

El **proceso de enfriamiento** es el mecanismo por el que la temperatura va tendiendo a 0. Es decir, la probabilidad de aceptación de soluciones peores tiende a 0. El proceso de enfriamiento, por tanto, determina cómo se modifica la temperatura durante la ejecución del algoritmo.

Hay dos modelos básicos que se pueden utilizar para el proceso de enfriamiento. En el primero podríamos ir enfriando muy lentamente con cada iteración de manera que en un número de pasos extremadamente alto la temperatura tienda a 0, ó utilizar un proceso de enfriamiento más rápido, pero antes de aplicar un cambio en la temperatura hay que dejar que el sistema llegue a un estado estacionario, es decir, un estado en el que se establezca la solución.

Algunos de los algoritmos de enfriamiento utilizados son los siguientes:

- Descensos constantes de temperaturas.
- Descenso exponencial: $t_{k+1} = \alpha t_k$ ($0,8 < \alpha < 0,99$)
- Criterio de Boltzmann: $t_k = t_0 / (1 + \log(k))$
- Esquema de Cauchy: $t_k = t_0 / (1 + k)$

Hay que intentar durante el proceso de enfriamiento que haya una alta aceptación al principio (de forma que se pueda realizar un proceso de exploración inicial en el espacio de búsqueda) e ir reduciendo esta aceptación hacia el final (de forma que se fomente el proceso de explotación de vecindad para llegar a una solución lo mejor posible).

El **número de repeticiones** del bucle interno nos va a dar el número de vecinos que se visitan antes de reducir la temperatura del

sistema. Es proceso debe ser dinámico, de forma que se aumente el número de repeticiones según se reduce la temperatura. De esta forma se conseguirá explotar la vecindad hacia el final del proceso.

Un mecanismo habitual para conseguirlo es repetir el bucle hasta que se cumpla una de las siguientes condiciones:

- Se aceptan un cierto número de soluciones. Durante las primeras fases del proceso se aceptarán muchas soluciones mientras que en las últimas fases se aceptarán muy pocas. De esta forma según se desarrolla el algoritmo cada vez se analizan más vecinos.
- Se han visitado un cierto número de vecinos. De esta forma el bucle interno termina en algún momento, si se da el caso que se aceptan muy pocos vecinos, o ninguno, de acuerdo con la condición anterior.

Para que se pueda explorar con cierta completitud todos los vecinos el número de repeticiones debe ser del orden del tamaño de la vecindad.

La condición de parada nos indica cuando damos por terminado el algoritmo de cálculo. Hay que tener en cuenta que al reducirse la temperatura, la probabilidad de aceptar soluciones peores tiende a 0. Así mismo, la probabilidad de encontrar soluciones mejores también tiende a 0.

En este sentido las condiciones para terminar el algoritmo deberían tener en cuenta que ya se han realizado un determinado número de iteraciones sin mejorar la solución. La idea es evitar seguir insistiendo reiteradamente en la vecindad de una solución para la que ya se lleva tiempo intentado explotar.

La definición del modelo del problema determinará la estructura del espacio de soluciones y, por tanto, la forma en que se va a poder recorrer, los algoritmos que se pueden utilizar para crear soluciones, etc.

Resulta de gran importancia que el modelo elegido para crear el modelo del problema disponga, por comodidad algorítmica, de algunas propiedades como: el espacio de soluciones no genere soluciones degeneradas, no permita la generación de soluciones imposibles, la estructura del espacio de costos no sea excesivamente escarpado o llano, etc.

Lógicamente para cada problema hay que estudiar de forma diferenciada cómo generar este espacio de soluciones, en relación con la facilidad para generar una vecindad y el cálculo de la función de coste asociada a una determinada solución.

Dada una solución, hay que poder conseguir una nueva solución mediante un “pequeño” cambio en la solución original. Además la forma de generar una vecindad nos debe poder asegurar que a partir de una solución cualquiera se debe poder llegar a cualquier otra ya sea directa o indirectamente. Esta característica es vital para dar validez al proceso de convergencia del algoritmo.

Asimismo, la vecindad de una solución dada debería ser suficientemente pequeña como para poder realizar una búsqueda en pocas iteraciones, como suficientemente grande para generar mejoras sustanciales en pocos movimientos.

Desde el punto de vista computacional la generación de un vecino debería ser rápida, de forma que se puedan calcular un mayor número de vecinos por unidad de tiempo. Así mismo, sería muy apropiado que el vecino generado siempre sea una solución factible, lo que puede resultar complicado. Si no es factible conseguir vecinos factibles, se puede permitir que durante el proceso del algoritmo se generen soluciones imposibles, pero éstas se penalicen suficientemente como para que no se elijan más que de forma temporal.

La función de costo mide la bondad de una solución y está asociada a todos los elementos de dicha solución. La función de costo debe ser rápida de calcular al generar una solución de vecindad, creando un cálculo incremental que considere solamente los cambios que genera la vecindad. De esta forma se podrán calcular más soluciones por unidad de tiempo. De hecho, el cálculo de la función de coste

suele ser uno de los aspectos más costosos en tiempo de cómputo del algoritmo.

Asimismo hay que evitar generar una topografía muy plana o muy abrupta. Si la topografía es muy plana, al generar vecinos el coste no variará para muchos de ellos, no permitiendo diferenciar soluciones. En el caso contrario, si la topografía es muy abrupta, la diferencia de coste entre una solución y un vecino de la misma será demasiado grande no permitiendo guiar el proceso de convergencia hacia una solución óptima.

La solución inicial, en general es cualquier solución válida del problema. Al ser un problema de optimización combinatoria, el modelo permite expresar una solución como una combinación de partes del mismo. De esta forma, la solución inicial más sencilla resulta de elegir una combinación aleatoria de elementos de problema que conformen una solución con la estructura del modelo elegido.

Otra alternativa es utilizar un proceso doble. En primer lugar se emplea algún tipo de heurística que nos permita generar una solución razonablemente buena, mejor que utilizar una solución estrictamente aleatoria.

3.2.3.2. Aplicaciones del Recocido Simulado al problema de la generación de horarios

La técnica de Recocido Simulado se ha estado utilizado en numerosos problemas de timetabling como el presentado por Mushi, A. R.[31] el cual trata el tema específico de programación de horarios para exámenes a nivel universitario, el cual involucra la asignación de los cursos del examen, las aulas, los horarios y los candidatos a dar examen. Para esto utiliza una matriz que permite detectar los conflictos entre exámenes.

La matriz $M_{n \times n}$ donde n representa el total de exámenes y $m_{ij} = 1$ si el examen i entra en conflicto con el examen j con relación a los candidatos y 0 en otro caso. La solución es representada por s_i el

horario asignado al examen i y k_i la lista de salones asignados al examen i . La función objetivo es una combinación lineal con pesos de todas las restricciones del problema, aunque usa dos funciones separadas una para las restricciones obligatorias y otra para las deseables logrando así un mejor desempeño.

$$f(x) = \sum_{i=1}^h \lambda_i f_i(x)$$

Donde f_i es la función a minimizar asociada a la restricción i y λ_i es el peso para esta restricción y h el número de restricciones. El peso de la restricción va de acuerdo a su importancia.

Para la generación de la solución inicial usan un proceso que la crea con la menor cantidad de conflictos posibles para agilizar el proceso de búsqueda, como en la función objetivo se definen todas las restricciones no es necesario partir con una solución factible.

Para escoger el criterio de enfriamiento se usa la función más conocida (método geométrico)

$$\alpha(t) = at, a < 1$$

Pero también como fines de prueba y comparación utiliza otra función la cual demora más en enfriar la temperatura.

$$\alpha(t) = t / (1 + at), a < 1$$

En cuanto a los movimientos en la vecindad de cada solución usa dos métodos para su generación:

Cambiando el tiempo del examen aleatoriamente de la siguiente forma:

- Seleccionamos un examen al azar
- Buscamos otro horario al azar
- Asignamos el horario al examen

Cambiando el horario mediante intercambio de dos horarios aleatorios:

- Seleccionar dos exámenes
- Seleccionar un horario
- Reemplazamos el horario del primer examen por el horario encontrado
- Reemplazar los horarios del primer examen con el segundo

Cabe destacar que este segundo métodos genera una vecindad mayor al primer método.

Como conclusiones al trabajo es que al probar descubrieron que los resultados alcanzaron alta desempeño tanto en tiempo y cumplimiento de las restricciones. Además se mostró que el enfriamiento geométrico y el segundo método para generar una solución vecina son los mejores para este problema.

Capítulo 4 : Aporte Teórico

En este capítulo se presentarán los fundamentos necesarios que darán solución al problema de generación de horarios a través de la aplicación de los algoritmos genéticos, así mismo delimitaremos de manera explícita el nuevo conocimiento del objeto de estudio.

4.1. Introducción

En este capítulo se pretende mostrar, en primer lugar, la justificación del uso de los algoritmos genéticos para resolver los problemas de timetabling. Se comenzará con una evaluación subjetiva en base a una serie de criterios que una heurística debe garantizar para ser considerada como buena, y luego se presentarán dos evaluaciones más cuantificables para comparar los algoritmos genéticos con otros métodos de optimización. La primera de estas evaluaciones pretende evaluar, bajo tiempos iguales de procesamiento, la calidad de las soluciones que se obtienen y la segunda evaluación muestra el tiempo requerido por los diferentes métodos para encontrar una solución factible al problema.

La segunda parte del capítulo nos describe la implementación modificada del Algoritmo Genético que es usada para solucionar nuestro problema de timetabling. Se presentará un estudio por cada elemento del algoritmo, los cuales se han definido en un capítulo anterior, explicando el método seleccionado y el por qué de su uso.

Un punto importante, que se debe observar, es el uso de una variación del método “divide y vencerás”, el cual se usará para mejorar el funcionamiento de la heurística. A través de esto se mostrará que el problema de asignación de recursos a cada una de las clases se puede solucionar en tres fases: primero, asignando docentes, luego asignando los horarios o periodos y por último asignando las aulas. Esto es posible sólo si se toman en cuenta restricciones que están relacionadas a cada fase y que son indispensables a su correspondiente fase anterior; por ejemplo, cualquier requisito acerca de las aulas tales como la cantidad máxima o que una clase necesita un aula con ciertas características se puede traducir a restricciones sobre los horarios, tal que después de asignar éstos, las aulas se pueden asignar sin ningún problema.

4.2. Evaluación y Selección del Algoritmo

Para realizar la implementación de la solución, primero se hizo un estudio de los diferentes algoritmos con los cuales se podría dar solución al problema, por lo que se procedió a realizar una comparación entre las diferentes alternativas detalladas en el capítulo anterior para sustentar nuestra elección.

4.2.1. Cuadro comparativo de Metaheurísticas

En este punto se realizará la comparación entre las metaheurísticas vistas, según las “buenas propiedades” de cada una de éstas. Primero una pequeña definición de cada una de estas características deseables [30]:

- Simplicidad, ya que una metaheurística debe estar basada en un principio sencillo y claro que la haga fácil de comprender,
- Independencia, puesto que no puede depender del marco ó del agente tecnológico en el que se vaya a desarrollar.

- Coherencia, porque los elementos que caractericen la metaheurística deben derivarse de forma natural de los principios que la inspiran.
- Efectividad, ya que los algoritmos particulares que se produzcan a partir de las metaheurísticas deben proporcionar soluciones óptimas ó muy cercanas a las óptimas, es decir, de alta calidad en algún sentido propio que cada usuario deberá especificar, y que podrá depender de cada problema concreto.
- Eficacia, en el sentido de fallar sólo en ocasiones muy raras ante casos prácticos del mundo real.
- Eficiencia, como una característica, más que deseable, exigible en términos de recursos, es decir, de tiempo de ejecución, de espacio de memoria y, en definitiva, de costos de desarrollo.
- Generalidad, de forma que se pueda utilizar provechosamente en una gama de situaciones y problemas tan amplia como sea posible.
- Adaptabilidad, para que pueda adecuarse a los diferentes contextos de aplicación y a los distintos casos que se consideran.
- Robustez, ya que no puede ser muy sensible a pequeñas alteraciones del modelo ó contexto de aplicación.
- Interactividad, para que el decisor pueda mejorarla a partir de su experiencia y conocimientos.
- Diversidad, para que se permita al usuario elegir entre las distintas soluciones alternativas que proporcione la metaheurística.
- Autonomía, para facilitar su funcionamiento automático global, ó al menos en alguna de las facetas que la caractericen.

Aquí mostraremos la comparación de los diferentes métodos para luego escoger entre uno de ellos. La calificación colocada a cada uno de ellos viene dada por la siguiente tabla:

Puntaje	Valor
0	Muy malo
0.25	Malo
0.5	Regular
0.75	Bueno
1	Muy bueno

Tabla 4.1. Puntuaciones para calificar las características de las metaheurísticas

Una vez definidos estos puntajes, procedemos a realizar la respectiva comparación:

Característica	Algoritmo Memético	Algoritmo Genético	Búsqueda Tabú	Recocido Simulado
Simplicidad	0.5	1	0.75	0.75
Independencia	1	1	1	1
Coherencia	0.75	1	0.75	1
Efectividad	1	1	1	0.75
Eficacia	0.75	0.75	0.75	0.5
Eficiencia	1	1	0.75	0.5
Generalidad	0.5	0.75	1	1
Adaptabilidad	1	1	1	0.75
Robustez	0.75	0.75	0.75	0.75
Interactividad	0.75	0.75	0.75	0.75
Diversidad	1	1	0.5	0.5
Autonomía	1	1	1	1
Puntaje Final	10	11	10	9.25

Tabla 4.2 Resultados de la calificación de las metaheurísticas

Por la tabla observamos que los Algoritmos Genéticos tienen una ventaja sobre el resto de meta heurísticas.

4.2.2. Análisis de tiempo y calidad de solución

Una vez realizada la comparación cualitativa entre los diferentes métodos, nos toca realizar una comparación cuantificable de acuerdo a la calidad de la solución.

Dado que los resultados de las mediciones del tiempo de término de procesamiento para cada método son diferentes, la evaluación, solamente, del valor final de cada solución no sería suficiente. Una alternativa es observar el comportamiento del tiempo de operación del algoritmo tal como lo muestra la imagen a continuación.

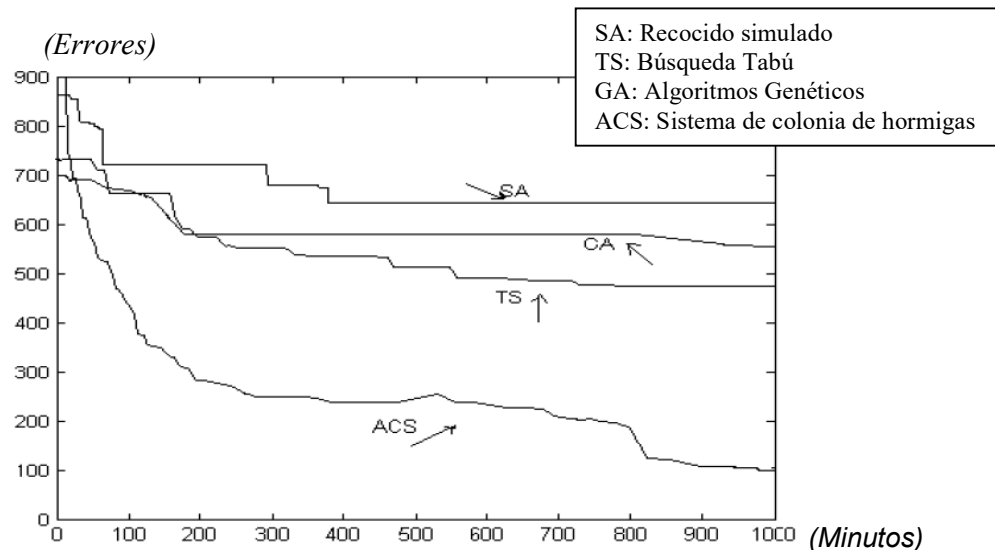


Figura 4.1. Comportamiento del tiempo de operación de las metaheurísticas

Pero según Zahra Naji Azimi [32], una de las estrategias que se puede adoptar es el uso de comparaciones dinámicas de los algoritmos en diferentes puntos, para ver el estado de sus soluciones en tiempos de procesamiento semejantes.

Según el trabajo de Zahra, quien realiza las pruebas para el problema de timetabling de exámenes, se obtuvo las siguientes mediciones.

P		Initial	Min 1		Min 2		Min 3		Cost Reduction
s1	Time		50	140		180			
	SA	938.5	881.9	0.47	813.7	1.03	765.5*	1.16	173.0
	TS	877.0	732.1	0.22	532.6*	0.33	532.6	0.50	344.3
	GA	771.8	668.2	0.12	627.4*	0.57	627.4	0.77	144.3
	ACS	811.0	597.8		399.8		354.4*		456.5
s2	Time		30	120		480			
	SA	1116.4	1100.2	0.05	1038.1	0.14	976.0*	0.25	140.3
	TS	1197.5	1066.8	0.01	912.3		835.0*	0.07	362.4
	GA	1226.2	1051.6		1016.5	0.1	975.7*	0.25	250.4
	ACS	1222.7	1171.2	0.11	1073.6	0.18	780.3*		442.3
s3	Time		60	300		540			
	SA	1217.3	1217.3	0.24	1150.3	0.24	1059.2*	0.19	158.1
	TS	1365.0	1802.8	0.84	956.0	0.03	905.3*	0.02	459.6
	GA	1149.3	1072.3	0.10	1072.3	0.16	965.5*	0.09	183.7
	ACS	1139.4	977.4		927.8		887.1*		252.2
s4	Time		60	300		540			
	SA	940.2	723.8	0.14	697.2	0.09	676.2*	0.15	263.9
	TS	750.8	697.1	0.09	646.0	0.01	585.7*		165.0
	GA	727.6	641.2*		641.2		641.2	0.09	86.3
	ACS	683.0	667.4	0.04	656.3	0.02	630.3*	0.08	52.7
s5	Time		100	300		540			
	SA	1015.2	994.5	0.10	979.9*	0.09	979.9	0.09	35.2
	TS	1125.5	1001.8	0.11	952.2	0.05	915.0*	0.01	210.4
	GA	986.6	902.4*		902.4		902.4		84.2
	ACS	1034.3	945.5	0.05	945.5	0.05	919.4*	0.02	114.8

SA: Recocido simulado
 TS: Búsqueda Tabú
 GA: Algoritmos Genéticos
 ACS: Sistema de colonia de hormigas

Tabla 4.3. Medición de soluciones en intervalos de tiempo

En esta tabla, el símbolo asterisco indica cuál metaheurística tiene el mínimo valor después de cada iteración. La mejor solución de cada metaheurística en el mismo tiempo y la que obtuvo la mejor reducción de costo está en negrita. La columna a la derecha de cada celda contiene la diferencia relativa con respecto a la mejor solución en el mismo tiempo.

$$Diferencia = \frac{(Costo_de_la_solucion - Costo_de_la_mejor_solucion)}{Costo_de_la_mejor_solucion}$$

Como resultado de estas mediciones se observa que los Algoritmos Genéticos obtuvieron una buena posición con respecto a los demás, sólo fue superado por el Sistema de Colonia de Hormigas, seguido

por la Búsqueda Tabú y el Recocido Simulado en relación a la búsqueda de la mejor solución en el mismo tiempo.

Algoritmo	SA	TS	GA	ACS
Habilidad de encontrar la mejor solución	3.3%	26.6%	26.6%	43.3%

Tabla 4.4. Habilidad de las meta heurísticas de encontrar la mejor solución

Los valores mostrados fueron calculados a través de la siguiente fórmula:

$$\text{porcentaje} = \frac{(\text{Puntos en que la heurística encontró la mejor solución})}{\text{Total de puntos}} * 100$$

El efecto de las cuatro metaheurísticas con relación a la solución inicial es diferente, lo cual se prueba al realizar el cálculo de la tasa de reducción para cada uno de los métodos. Los Algoritmos Genéticos poseen un costo de reducción bajo de la población pues mantienen más amplio el espacio de soluciones, permitiendo así no caer fácilmente en óptimos locales.

Algoritmo	SA	TS	GA	ACS
Habilidad de producir un máximo costo de reducción	10%	60%	10%	20%

Tabla 4.5. Habilidad de las metaheurísticas de producir un máximo costo de reducción

4.2.3. Comparación de tiempos

En cuanto a la comparación de tiempos que demoran en converger cada uno de los métodos hacia una buena solución, tenemos el trabajo realizado por Silvia Casado Yusta y Joaquín Pacheco Bonrostro [26]. En dicho trabajo se prueban diferentes metaheurísticas bajo características comunes, realizando la medición del tiempo en el cual se hallaba la mejor solución por cada algoritmo

o si se alcanzaba la condición de parada, obteniendo los siguientes resultados.

	GRASP	VNS	TS	AG	AM	BT
Valor	3291	3319	3311	3290	3281	3274
<i>t.total.</i>	828,07	207,03	279,43	1702,00	13711,77	960,24
<i>t.mej.sol.</i>	159,57	81,32	161,63	588,31	8606,87	148,87

GRASP: Algoritmo GRASP
 VNS: Búsqueda en entorno de variables
 TS: Recocido simulado
 AG: Algoritmos Genéticos
 AM: Algoritmos Meméticos
 BT: Búsqueda Tabú

Tabla 4.6. Comparación de tiempos para alcanzar una adecuada solución

De la Tabla 4.6 se obtienen los siguientes resultados:

- Se observa cómo la estrategia que nos permite obtener los mejores resultados en un tiempo de computación corto es Búsqueda Tabú. Pero también dan buenos resultados el Algoritmo Memético y el Algoritmo Genético y en menor medida el Algoritmo GRASP. En cuanto a Recocido Simulado y Búsqueda en Entornos Variables se obtienen buenos resultados rápidamente, pero no muestran capacidad de mejorarlos posteriormente.
- Los Algoritmos Genéticos y Memético dan soluciones bastante cercanas al óptimo que Búsqueda Tabú, además emplean un tiempo de computación mucho mayor que el resto de las estrategias para alcanzar estas soluciones. Llama la atención especialmente el tiempo de computación empleado por el Algoritmo Memético (8606,87 segundos en alcanzar la mejor solución).

De lo evaluado se podría considerar a los algoritmos genéticos como buena opción para dar solución al problema de timetabling, si bien tiene varios puntos a favor comparado con otras heurísticas al

momento de resolver problemas de timeabling, también podemos ver que posee desventajas que trataremos de eliminar con su respectiva adaptación a nuestro problema.

4.3. Adaptación del algoritmo

Tomando en cuenta las bases teóricas mostradas en los capítulos 2 y 3, la estructura del problema observado en el capítulo 1 y considerando los motivos por los cuales se seleccionó los Algoritmos Genéticos, la parte de este capítulo tiene como objetivo diseñar un algoritmo que trate con eficacia y eficiencia el problema. Las modificaciones que se realicen al algoritmo se tratarán en los siguientes puntos.

4.3.1. Unidad de asignación

Podríamos interpretar el problema de timetabling como el problema de asignar un conjunto de recursos a un elemento en particular, como por ejemplo la asignación de docentes, de horarios y aulas a un curso específico en su representación más sencilla.

En nuestro caso el elemento al cual le vamos a asignar los recursos (docentes, horarios y aulas) es la clase; la cual se define como la combinación de un curso, el tipo de éste, la cantidad de horas a dictar y el grupo.

CURSO	TIPO DE CURSO	HORAS DE DICTADO	GRUPO
-------	---------------	------------------	-------

Figura 4.2. Estructura de la unidad de asignación (clase)

Una vez definido esto, nuestro algoritmo se encargará de asignar un docente, un horario y un aula determinada a cada una de las clases, los cuales son necesarios para llevarla a cabo.

El proceso que generará el universo de clases, que servirán de elementos de entrada al algoritmo, será un procedimiento lineal. Este tomará en cuenta los tipos de curso (teoría, práctica y laboratorio), la cantidad de horas de dictado por cada tipo, los grupos que se aperturen para el semestre en curso y la siguiente restricción obligatoria:

“**ROG7**. Un curso debe cumplir con una cantidad de horas semanales requeridas. Según sea el caso si el número de horas académicas es par se asignará dos horas diarias hasta cubrir la totalidad, sino se asignará un día de tres horas y el resto de dos horas. A cada uno de estos grupos de horas se les llamará bloques”.

Definiendo de este modo la unidad a la cual se asignará los recursos y asegurando que el algoritmo asignará recursos a todas las clases sin excepción con la finalidad de generar un horario factible, entonces se tendrán cumplidas de antemano la restricción obligatoria anterior así como la siguiente:

“**ROG8**. Las horas de un bloque para un determinado curso en un mismo día deben de ser consecutivas además de ser asignados a un mismo docente y aula”

Esto nos permitirá ahorrar tiempo computacional debido a que se evitará la evaluación del cumplimiento de estas restricciones, y además se podrá simplificar la implementación del algoritmo.

Mediante un ejemplo sencillo veremos la generación de las clases y su correspondiente estructura: Consideremos el curso de Algorítmica I, el cual tiene 3 tipos definidos y se le aperturarán 3 grupos en el presente semestre, además en la siguiente tabla tenemos las horas semanales de dictado por cada tipo definido para el curso:

Tipo de Curso	Horas de Dictado
Teoría	5
Practica	3
Laboratorio	4

Tabla 4.7. Horas de Dictado de Algorítmica I

Ahora tomaremos en cuenta lo expresado en la restricción **ROG7** : Para el curso de Algorítmica I de tipo teoría se tiene asociado una cantidad de horas de dictado impar, por lo tanto se tendría una clase de teoría de tres horas y otra de dos. Por consiguiente el total de clases generadas serían:

Curso	Tipo	Horas	Grupo
Algorítmica I	Teoría	3	1
Algorítmica I	Teoría	2	1
Algorítmica I	Practica	3	1
Algorítmica I	Laboratorio	2	1
Algorítmica I	Laboratorio	2	1
Algorítmica I	Teoría	2	2
Algorítmica I	Teoría	3	2
Algorítmica I	Practica	3	2
Algorítmica I	Laboratorio	2	2
Algorítmica I	Laboratorio	2	2
Algorítmica I	Teoría	2	3
Algorítmica I	Teoría	3	3
Algorítmica I	Practica	3	3
Algorítmica I	Laboratorio	2	3
Algorítmica I	Laboratorio	2	3

Tabla 4.8. Clases generadas para el curso de Algorítmica I

4.3.2. Separación en Sub-Fases

En este punto veremos cómo el problema, que como ya hemos mencionado anteriormente corresponde a la asignación de recursos (docentes, aulas, horarios) a las clases, será simplificado debido a que no se asignarán todos en un mismo momento. Dicha asignación se realizará uno a uno a través de diferentes fases delimitándose el espacio de soluciones cada vez que se van asignando dichos recursos.

A continuación definiremos 3 fases según los recursos a asignar:

- Fase 1: Asignación de docentes a las clases.
- Fase 2: Asignación de horarios a las clases.
- Fase 3: Asignación de aulas a las clases.

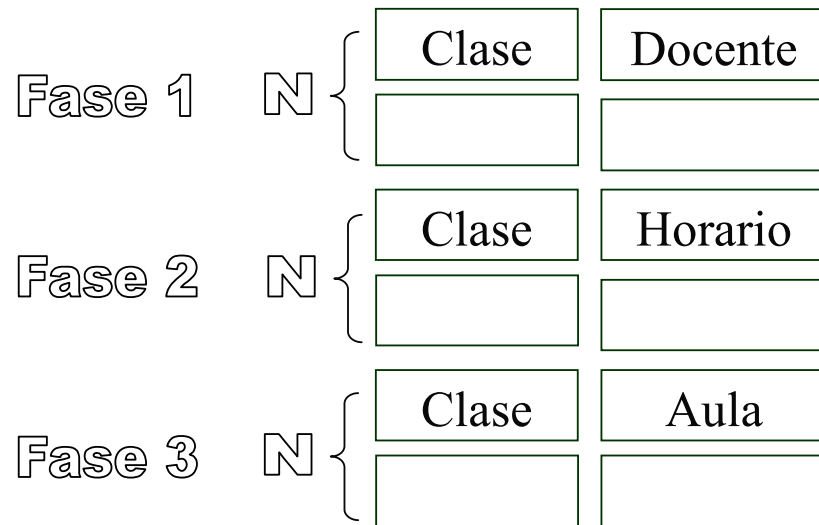


Figura 4.3. Fases para la asignación de clases a los recursos

A continuación veremos con más detalle cada una de las fases y la forma como éstas interactúan:

La **primera fase** asigna los docentes a las clases, tomando en cuenta que todas las restricciones obligatorias correspondientes al docente y al curso se cumplan. La solución resultante pasará a la **segunda fase**, la cual se encargará de asignar diversos horarios a las clases cerciorándose de que se cumplan las restricciones relacionadas a los periodos de tiempo, cursos y docentes; además se debe cerciorar que cada horario no tenga más asignaciones que aulas disponibles, dado de que si esto no se verifica entonces no existiría solución posible para la siguiente fase. La **tercera fase** asigna aulas a las clases, las cuales ya poseen horarios y docentes asignados como resultado de la primera y segunda fase.

Puesto que existe un mayor número de restricciones implicadas con los horarios, resulta mejor considerar esta fase en segundo lugar debido a que controlaremos las restricciones de los docentes con respecto a los horarios que no se pudieron controlar en la primera fase. Y no se ha considerado como última fase debido a que no hay restricciones en el uso de las aulas con respecto a los horarios considerados.

En la fase uno encontramos restricciones que están relacionadas con la preferencia de dictado de cursos por parte de los docentes y la disponibilidad de éstos. En la fase dos, referida a los horarios, hay restricciones tales como la determinación de la facultad a programar algunos cursos solamente por la mañana ó por la tarde, la no posibilidad de programarlos en ciertas horas, entre otras. En la fase tres podemos manejar restricciones relacionadas a la necesidad de ciertas clases por hacer uso de aulas específicas. Esto puede ser ejemplificado con la programación de aulas para las clases de laboratorio de ciertos cursos. Sin embargo, debemos observar que hay relativamente pocas restricciones referidas a las asignaciones de las aulas porque el número de aulas en la facultad es grande. De hecho, la asignación de aulas a las clases que tienen ya horarios asignados es una tarea de menor complejidad comparada a la de asignar horarios a las clases, igualmente ocurre con la primera fase correspondiente a la asignación de docentes la cual también es de menor complejidad.

Ahora veremos la justificación de separar el problema en fases. Si quisiéramos atacar el problema en una sola fase implicaría validar todas las restricciones obligatorias por la asignación de horarios, docentes y aulas a las clases, tendríamos entonces restricciones obligatorias tales como que no se debe asignar dos o más clases en el mismo horario y aula, ó asignar a un docente a más de una clase en el mismo horario. Puesto que un algoritmo genético es una técnica metaheurística, esto es absolutamente posible a lo largo del curso de su evolución. Dado que penalizamos tal situación, los algoritmos genéticos se esfuerzan eventualmente a quitar tales asignaciones, pero esto está a expensas de, posiblemente, cientos de iteraciones para encontrar la solución factible. Para ilustrar esta idea supongamos que la asignación del docente y horario ya están definidos y ocurriese un conflicto de dos clases en un mismo horario y aula, simplemente deberíamos cambiar el aula de clases a otra aula con el mismo horario; esto es cierto especialmente para esas clases que no tienen preferencia alguna de lugar. Este cambio es siempre posible, y se hace a menudo, siempre y cuando el número de las clases asignadas a un horario no exceda el número de las aulas disponibles en ese horario. Lo mismo ocurre con los docentes, podemos cambiar el horario asignado a otro horario con el mismo docente siempre y cuando los horarios disponibles para el docente no sean sobrepasados.

Actualmente, el número de aulas disponibles es el mismo en cualquier horario, no cambia con respecto al tiempo, en otras palabras las aulas se encuentran disponibles en su totalidad para todos los horarios considerados; mientras que para el caso de los docentes y su disponibilidad, éstos son considerados datos más volátiles y complejos. Por lo tanto, podemos ahorrar centenares de iteraciones adicionales de heurística, separando el problema en fases.

La simplificación y la solución del problema, dividiéndolo en estas tres fases fomentan el hallazgo de soluciones de una manera más rápida, puesto que la combinación de los conflictos de los docentes, de los horarios y de las aulas no se considera simultáneamente, sino en diversas fases.

4.3.3. Representación del problema

Dado que hemos mencionado la existencia de tres fases diferentes para dar solución al problema tendremos que definir tres cromosomas, uno para cada fase. Cada uno de los cromosomas tendrá una cantidad de genes igual al número de clases y el valor de estos genes (alelos) será igual al recurso asignado a la clase. Una ilustración áspera de un cromosoma es como sigue:

Clase 1	Clase 2	Clase 3	...	Clase N
Recurso 2	Recurso 20	Recurso m	...	Recurso 9

Figura 4.4. Representación vectorial del cromosoma

Es importante observar que los tres cromosomas son iguales a nivel de estructura, pero aquello que los diferenciará será el significado que tengan los alelos (valores de los genes) en cada fase; es decir, los alfabetos usados en cada una de las fases serán distintos.

4.3.4. Codificación mediante alfabetos duros

La elección del tipo de codificación de las soluciones influye en los tipos de operadores que se pueden utilizar, así que este proceso debe ser realizado cuidadosamente tratando que se refleje en la medida de lo posible la naturaleza del problema.

Como ya se ha mostrado teóricamente existe una codificación directa y una codificación indirecta, siendo ésta última la más común. En la codificación indirecta se traza la gama de valores posibles en una dimensión más simple, a menudo un espacio binario. La representación binaria tiene ventajas de cómputo y de programación debido a que presenta una estructura más simple; sin embargo, cuando tratamos casos más complejos es difícil tratarlo a través de una codificación indirecta debido a la complejidad de cómputo que implicaría tener que traducir cada representación.

Para nuestro caso, notamos que la utilización de una representación directa, debido a su facilidad de evaluación, permite el diseño de un sistema más complejo tanto en sus condiciones y sus objetivos. Es por ello que para nuestro modelo consideraremos una codificación directa en lugar de una codificación indirecta.

Ahora que ya establecimos la estructura y la codificación a utilizar, pasaremos a mostrar el alfabeto que se empleará, en base a los requisitos particulares, en cada una de las fases: En la fase uno, el alfabeto constará de enteros que representarán a cada docente, los cuales serán asignados a cada una de las clases en el semestre actual. Con esto tendremos un alfabeto específico asociado a cada clase a la cual le vayamos asignar un docente. Para las fases dos y tres se usará la misma forma de representación tanto para los horarios como para las aulas a asignar a cada clase del semestre actual.

De esta manera logramos que el valor que vaya a tomar cada gen no se encuentre fuera del dominio establecido para cada recurso, ya que con una representación binaria podríamos obtener un conjunto de valores que no sean correctos y que no correspondan al rango de valores de los recursos. Por ejemplo, tenemos un conjunto de 20 docentes cuyo alfabeto estará definido desde el valor 1 al 20 y si para ello usamos una combinación binaria

para representar dichos valores, entonces podríamos obtener valores que se encuentren fuera de este rango.

También puede suceder que a las clases no se les pueda asignar ciertas aulas o ciertos horarios o ciertos docentes por algún motivo, entonces nosotros podemos quitar esas combinaciones del alfabeto correspondiente. Por ejemplo, en caso que queramos asignar un docente a una clase, deberíamos primero seleccionar al docente entre los que ya están definidos; pero imaginemos que sólo dos de ellos tienen preferencia para el dictado de dicha clase, entonces si quitamos ciertos valores del alfabeto evitaríamos que se asigne un docente que no tenga preferencia por dicha clase.

Esta clase de codificación dura protege al modelo contra la violación de restricciones obligatorias, ahorrando de esta manera recursos computacionales y minimizando el tiempo. Bajo este criterio es imposible que los segmentos del cromosoma tomen valores fuera del espacio factible de soluciones. En nuestro problema timetabling, esto significa que si un docente desea enseñar solamente en los horarios de la mañana, el algoritmo genético nunca le asignará una clase de la tarde en toda su población, desde la primera iteración hasta la última. La probabilidad de que no se cumpla esta restricción es nula; sin embargo, esto también hace que al sistema le resulte difícil encontrar una solución, puesto que se ocupará de generar los diversos espacios de alfabetos que estarán asociados a las clases en cada una de sus fases.

Una vez detallada la conceptualización de nuestra idea veamos las restricciones que se emplearán en cada una de las fases, las cuales limitarán el uso de alfabetos en los diferentes genes de la población y a su vez no tendrán que ser tomadas en cuenta en la evaluación de la solución. En la fase uno (docente - clase), acotar la lista de docentes que puedan dictar una clase determinada, nos permitirá dejar de lado la evaluación de las siguientes restricciones:

- **“ROG4.** No es posible asignar a un docente un curso por el cual no posea preferencia.”
- **“ROG5.** No se puede asignar un curso a un docente fuera de su disponibilidad de tiempo.”

- **“ROE6.** Pueden existir asignaciones previas de docentes a un determinado periodo y curso”
- **“ROE7.** Los cursos son clasificados en los turnos de mañana, tarde y noche según el semestre al que pertenecen”
- **“ROE5.** Pueden existir asignaciones previas de cursos a un determinado periodo y aula”

En la fase dos (horario - clase), análogamente a lo indicado en la fase uno, los horarios en los que se pueda dictar una clase, nos permitirá dejar de lado la evaluación de las siguientes restricciones:

- **“ROE5.** Pueden existir asignaciones previas de cursos, a un determinado periodo y aula”
- **“ROE7.** Los cursos son clasificados en los turnos de mañana, tarde y noche según el semestre al que pertenecen”
- **“ROG5.** No se puede asignar un curso a un docente fuera de su disponibilidad de tiempo.”

Y por último en la fase tres (aula - clase), la limitación de las aulas en las que se pueda dictar una clase, nos permitirá dejar de lado la evaluación de las siguientes restricciones:

- **“ROE2.** Las clases de teoría y práctica deben de asignarse a aulas de tipo no laboratorio.”
- **“ROE3.** Los cursos de laboratorio deben de asignarse a aulas de tipo laboratorio.”
- **“ROG6.** La cantidad de estudiantes en un curso no puede sobrepasar la capacidad del aula.”

4.3.5. Inicialización de la población

La primera etapa en un Algoritmo Genético es la generación de la población inicial, que como ya se mencionó se suele realizar de manera aleatoria. Aquí se darán valores a todos los genes de los N cromosomas, donde N representa el tamaño de la población inicial, es decir, la cantidad de soluciones que se considerarán en cada etapa.

Como en nuestro caso se abordará el problema en tres fases diferentes teniendo para cada una de éstas un algoritmo genético asociado, el cual se encargará de la asignación de un recurso en particular (docentes, horarios y aulas) a las clases, necesitaremos entonces también tres fases de inicialización.

Con respecto a la estructura del cromosoma, en nuestro caso estará determinada por un arreglo de enteros para cada una de las tres fases consideradas. Una primera apreciación podría considerar la asignación, a cada índice del arreglo, de valores aleatorios dentro del conjunto de los enteros, desde 1 a un límite superior que vendría a ser el número total del recurso según la fase. Por ejemplo consideremos que deseamos inicializar diez clases con docentes y contamos con una plana de veinte docentes, entonces tendremos un arreglo de longitud diez donde cada índice del arreglo corresponderá a una clase, y el arreglo como sabemos representará al cromosoma. Además codificaremos cada docente del uno al veinte de manera consecutiva. Por lo tanto la inicialización comenzaría seleccionando un valor aleatorio entre uno y veinte para asignarlo al índice 0 del arreglo, luego escogeríamos otro valor aleatorio en el mismo rango y lo asignaríamos al valor del índice 1, y así sucesivamente hasta completar el cromosoma con valores y posteriormente formar todos los cromosomas de la población inicial.

Con esto tendríamos la población lista para comenzar a ejecutar el algoritmo, sin embargo vemos ciertas deficiencias en este método.

Imaginemos que queremos asignar un docente a la clase de Algorítmica I y según el método de inicialización descrito, deberíamos entonces asignarle aleatoriamente un docente entre los veinte ya definidos, Resulta que sólo tres de ellos tienen preferencia para el dictado de Algorítmica I, entonces podría suceder que se asigne en la población inicial un docente que no tenga preferencia por el curso. Las opciones para el manejo de este error serían: Desecharlas, asignándoles una baja aptitud en el momento de la evaluación, lo cual implicaría su desaparición en términos estadísticos al cabo de unas pocas generaciones junto a la pérdida de información valiosa que pudieran contener; o Repararlas, lo cual aparece como una tarea costosa pues habrá de ser realizada en una gran cantidad de ocasiones a lo

largo del proceso evolutivo. Ninguna de las alternativas parece óptima en sí misma.

La opción que proponemos consiste en evitar la generación de este tipo de individuos no factibles. Este problema podría evitarse si al momento de generar la población inicial no se tomarán en cuenta ciertos recursos que por las restricciones previamente nombradas, sería imposible que se use para una determinada clase. Usando el mismo ejemplo anterior para la clase de Algorítmica I debería asignarse un docente aleatorio pero no del conjunto total de veinte docentes sino solamente de los tres posibles que pueden dictar dicho curso.

Con esto estamos generando una población inicial que de antemano ya cumple ciertas restricciones, lo cual aliviará la tarea del método que evaluará las soluciones, ya que no serán tomados en cuenta durante la evaluación. Dado que si los cromosomas iniciales no consideran estos valores, las poblaciones subsecuentes generadas nunca los tendrán presentes, con ello se pretende ahorrar tiempo en la ejecución del algoritmo. Esta solución tiene la ventaja añadida de permitir eliminar directamente soluciones de muy mala calidad sin necesidad de evaluarlas.

Dado que la estructura del cromosoma es igual en cada fase, esto es, un arreglo de longitud igual al número de clases a las cuales se les va asignar un recurso, el método utilizado (mostrado en el ejemplo anterior) para la inicialización de la población será el mismo para cada una de las fases. Lo que variará por cada fase, aparte del recurso en particular, serán los alfabetos generados para cada gen de los cromosomas; esto se realizará a través de un procedimiento lineal que acotará dichos alfabetos.

El hacer uso de un alfabeto en particular para cada gen del cromosoma, en cualquiera de las tres fases, nos permitirá en gran medida ahorrar la evaluación de un gran número de restricciones, dado que las soluciones iniciales elaboradas no presentarán la infracción de estas restricciones. Con lo cual podríamos considerar ésta como una mejor solución que simplemente el hecho de asignar valores aleatorios.

Otro aspecto a tener en cuenta, es la posibilidad de introducir en la población inicial soluciones de buena calidad, por ejemplo, procedentes de la experiencia acumulada en procesos anteriores. Esta forma de introducir

conocimiento específico es muy útil pues permite al algoritmo contar con información valiosa inicialmente. No obstante debe ser realizado con cuidado, pues sembrar la población inicial con soluciones no aleatorias puede alterar el proceso evolutivo al dirigir la búsqueda hacia un determinado espacio.

4.3.6. Función de evaluación y tratamientos de las restricciones

Como se pudo notar, la evaluación es la unión entre el Algoritmo Genético y el mundo externo. La evaluación se realiza a través de una función que representa de forma adecuada el problema y tiene como objetivo suministrar una medida de aptitud a cada individuo de la población actual. Para nuestro problema necesitaremos una función de evaluación durante las fases, las cuales considerarán las restricciones relacionadas con el recurso a asignar.

El modelo de evaluación de restricciones tendrá la estructura que se indicará a continuación. La función de evaluación tiene en cuenta el cumplimiento de las restricciones a través de una expresión ponderada de las mismas de acuerdo a:

$$f_{eval} = w_o \cdot \prod_{i=1}^n x_i + w_d \cdot \prod_{j=1}^m y_j$$

Donde $x_1 \dots x_n$ son las aptitudes calculadas de cada restricción obligatoria asociada a la fase y $y_1 \dots y_m$ son las aptitudes que corresponden al cumplimiento de las restricciones deseables de la misma fase.

Además w_o y w_d son los pesos o penalidades de las restricciones obligatorias y deseables respectivamente, cumpliéndose que:

$$w_o + w_d = 1$$

Dado que, una solución es considerada factible cuando se tiene como condición necesaria y suficiente que se debe verificar el cumplimiento de todas las restricciones obligatorias, sería razonable considerar que $w_o \gg w_d$.

De esta forma, una baja aptitud en el cumplimiento de las restricciones obligatorias penaliza de forma importante al individuo considerado.

Luego, las aptitudes calculadas de cada restricción x_i serán inversamente proporcionales al número de ocurrencias de la restricción obligatoria en el individuo a evaluar:

$$x_i = 1 / (1 + \text{total_de_violaciones_a_la_restriccion_obligatoria})$$

Y las aptitudes calculadas de cada restricción y_i serán inversamente proporcionales al número de ocurrencias de la restricción deseable en el individuo a evaluar:

$$y_i = 1 / (1 + \text{total_de_violaciones_a_la_restriccion_deseable})$$

Considerando el alcance del presente trabajo, no se tomará en cuenta las restricciones deseables, por lo cual el problema se reduce al retirarlas del cálculo de la aptitud de los individuos, asignándole de esta manera el valor 1 a w_o :

$$f_{eval} = \prod_{i=1}^n x_i$$

Ahora veremos las restricciones que serán tomadas en cuenta en cada fase para su respectiva evaluación en este problema.

Para la fase uno consideraremos las siguientes restricciones a evaluar:

- “**ROE1**. Un docente debe de impartir un número determinado de horas de dictado como máximo y como mínimo según su categoría”
- Además se agregará una serie de restricciones para garantizar que exista una solución factible para la siguiente fase del problema; las cuales serán: que no se sobrepase la disponibilidad máxima por curso de cada docente, que por lo menos exista una solución en la cual las clases de un mismo curso y grupo pero de diferentes tipos no se dicten el mismo día, y que por lo menos exista una solución en la cual los

cursos de un mismo semestre y grupo no se crucen debido a sus horarios de dictado.

Para la fase dos se considerarán las siguientes restricciones, las cuales pueden involucrar tanto a los horarios como a los docentes, dado que éstos últimos ya se encuentran asignados a la clase en la fase anterior:

- “**ROE4**. No deben coincidir los horarios de los cursos que corresponden a un mismo semestre y grupo.”
- Además se agrega una restricción más para garantizar que exista una solución factible para la siguiente fase del problema, ésta sería que no se asigne más clases en un mismo horario que el total de aulas disponibles para éste en la siguiente fase.

Para la fase tres se consideran las siguientes restricciones, las cuales pueden involucrar tanto a los horarios, a los docentes y a las aulas, dado que los dos primeros recursos ya se encuentran asignados a la clase en la fase anterior:

- “**ROG1**. Un curso debe de tener asignado a lo más un docente en un aula en un periodo específico”.
- “**ROG2**. Un docente debe tener asignado a lo más un curso en un aula en un periodo específico”.
- “**ROG3**. Un aula puede tener a lo más un curso asignado en un periodo específico”.
- “**ROG9**. Los cursos no deben de solaparse entre ellos, ni en duración ni ubicación física, una vez seleccionada la hora de inicio de éstos y el aula correspondiente.”

4.3.7. El proceso de selección

El proceso de selección en los algoritmos genéticos selecciona individuos para la reproducción. La selección está basada en la aptitud de los individuos: Individuos más aptos tienen mayor probabilidad de ser escogidos para la reproducción. Para nuestro caso en particular usaremos los mismos métodos de selección para las tres diferentes fases.

Uno de los problemas más comunes en un algoritmo genético es tener un súper-individuo al inicio de las iteraciones. Este súper-individuo tendrá una proporción equitativa a su aptitud para reproducirse, eliminando totalmente a los otros individuos y clonándose así en cada generación; haciendo de esta manera que el algoritmo genético converja prematuramente, tal vez en una solución no tan cercana a la óptima. Pero tampoco podemos dejar de proporcionar a los individuos más fuertes una mayor cantidad ocasiones de reproducción, de lo contrario reduciremos al algoritmo genético a una búsqueda puramente al azar. Hay varios métodos para solucionar este problema y dos procedimientos de uso muy frecuente, que serán tomados en cuenta para nuestra implementación.

4.3.7.1. Escalamiento de la aptitud

Antes del proceso de selección, los valores de la aptitud se pueden escalar en números de un rango más estrecho para hacer que las diferencias relativas lleguen a ser más pequeñas mientras que se mantienen las mismas posiciones. El resultado es que los súper-individuos tendrán mayor opción de reproducirse que otros individuos, pero no estarán dominando la población como lo harían sin un escalamiento de la aptitud. No escalaremos arbitrariamente los valores de la aptitud, la idea es asegurarnos que por lo menos un individuo medio antes del escalamiento permanezca siendo tal después del proceso de escalar. Un escalamiento simple pero bueno es el lineal: $f' = a + b*f$, donde el valor de la aptitud escalado es f' con a y b como operadores del escalamiento.

Para definir la escala adecuada a nuestro problema se escogió la fase dos (asignación de horarios a las clases), como la fase representativa para este proceso. Se realizaron las pruebas respectivas y se obtuvo como resultado que no es conveniente realizar el escalamiento de la función de aptitud. Dichas pruebas se encuentran descritas en el punto 6.3.1.1 del capítulo 6 denominado Pruebas y Resultados.

4.3.7.2. Selección de torneo

La selección de torneo es de desarrollo mucho más reciente en los algoritmos genéticos, e intenta tratar el problema de la convergencia prematura que se da debido a la existencia de los súper-individuos. Después de calcular los valores de las aptitudes, se selecciona un número de individuos aleatoriamente para un “torneo.” El individuo con la aptitud más alta en el torneo se selecciona para ser el padre. Cuando el número de individuos que se escoge para el torneo es más grande, es más probable la selección de las mejores soluciones; es decir, hay un mayor elitismo. Pero es relativamente más fácil que algunos individuos no muy aptos sean seleccionados en comparación con la selección de la rueda de ruleta descrita en los capítulos anteriores; esto es porque aunque exista un individuo dominante, éste no será seleccionado para la reproducción, si primero no es seleccionado a participar en el torneo. Por supuesto, ganaría cualquier torneo, pero todavía habrían torneos en los cuales no podría ser seleccionado, por lo tanto, es importante saber que para el torneo se seleccionan a los participantes aleatoriamente sin predisposición. Al mismo tiempo, este proceso de selección minimiza la presencia de individuos muy débiles, ya que es muy poco probable que ganen cualquier torneo; es por ello que la posibilidad de que la población esté integrada solamente de individuos débiles es casi inexistente. Resumiendo, la selección del torneo protege la diversidad de la población disminuyendo la opción de reproducción de los súper-individuos, y a su vez mantiene baja la de los individuos muy débiles. Tiene el efecto combinado del elitismo, porque descarta a individuos débiles, y diversificación, puesto que no se permite a los súper-individuos dominar.

Este método se relaciona al proceso evolutivo de una manera más aproximada, porque la supervivencia depende de la competencia entre pocos individuos más que de toda la población. También, la naturaleza no mata al más débil de los individuos, sino hace que los individuos más fuertes gobiernen sobre los otros. En este sentido, la reproducción de un súper-individuo no es conveniente, porque la poca diversidad puede matar fácilmente una población.

Para comprobar que este método se comporta mejor en nuestro problema, se realizaron las pruebas respectivas y los resultados que se obtuvieron están descritos en el punto 6.3.1.2 del capítulo 6.

Una vez optado por el método de torneo como solución para nuestro problema, ahora debemos identificar el tamaño del torneo. Para lo cual también se han realizado varias pruebas que están descritas en el punto 6.3.1.3 del capítulo 6, dando al final como resultado que el valor dos es el mejor tamaño que se adecua a nuestro problema.

4.3.8. Criterio de Término

Para la implementación de la solución se considerarán tres criterios de parada; cualquiera de los tres que se cumpla primero hará que el proceso del algoritmo culmine:

1. El primer criterio verificará si se ha alcanzado el número máximo de generaciones establecido, lo que evitará que se caiga en búsquedas demasiado largas y en las que tal vez no se alcance nunca una solución óptima.
2. El segundo criterio, es un término de parada por haber encontrado una solución factible al problema.
3. El tercer criterio detectará si todos los individuos de la población convergen en un óptimo local; es decir, si todos los individuos tienen en un determinado momento la misma aptitud.

Las tres fases consideradas tomarán en cuenta estos tres criterios, mas el primer criterio que corresponde al número de generaciones será uno distinto para cada fase, mientras que el segundo y tercer criterio serán los mismos para las tres fases.

El segundo criterio de parada es alcanzado cuando la función de evaluación toma el valor de 1, tal como se demostrará a continuación.

Dado que definimos nuestra función objetivo como:

$$f_{eval} = \prod_{i=1}^n x_i$$

Y x_i el resultado de:

$$x_i = 1 / (1 + total_de_violaciones_a_la_restriccion_obligatoria)$$

Se tiene que una restricción obligatoria está totalmente cumplida si $x_i = 1$, dado que hay 0 violaciones a la restricción. Por lo tanto, teniendo todas las restricciones satisfechas, la f_{eval} quedaría reducida a la multiplicación de “n” veces el valor de 1, siendo “n” el número de restricciones consideradas para la fase.

4.3.9. Operadores genéticos

En esta sección veremos los operadores genéticos considerados para nuestra solución, los cuales fueron probados para la segunda fase (asignación de horarios a las clases) dado que es la más compleja y representativa para nuestro problema.

4.3.9.1. Operador de emparejamiento

Como vimos en el capítulo anterior, la reproducción de la población queda a cargo de los operadores de emparejamiento. Estos operadores combinan los padres elegidos en el proceso de selección y generan una nueva solución; la cual tiende a ser más apropiada para el problema ya que fue generada por el cruzamiento de dos buenos individuos.

Para determinar cuál es el mejor operador de emparejamiento y cuál es el porcentaje de ocurrencia de dicho operador, se efectuaron un conjunto de pruebas que se encuentran detalladas en el punto 6.3.2.1 y 6.3.2.1.1 del capítulo 6. El resultado de dichas pruebas indica que el mejor es el operador de emparejamiento uniforme.

4.3.9.2. Operador de mutación

El empleo de un gran número de generaciones, independientemente del tamaño de la población inicial, puede derivar en el estancamiento del algoritmo dentro de un óptimo local (una solución que sólo satisface ciertas restricciones del problema). Para evitar este estancamiento utilizamos la mutación, la cual nos garantiza una mayor variedad de individuos y nos permite explorar mejor el espacio de soluciones posibles.

La mutación se puede realizar de varias maneras, pero dado que hacemos uso de un alfabeto no-contiguo de números enteros fijos, nos vemos en la necesidad de utilizar un operador modificado de la mutación para los requisitos particulares del problema. El operador de mutación debe modificar un valor por otro al azar, pero para nuestro caso de asignación de recursos a las clases se debe considerar que no se debe asignar cualquier valor a una clase en particular. Por ejemplo, para la fase uno que corresponde a la asignación de docentes a las clases, si tenemos que nuestra clase es Algorítmica I y el alfabeto está limitado por solamente tres docentes que enseñan dicho curso, entonces la asignación al azar se debe realizar considerando esos tres docentes (valores que pueden asignarse a dicha clase).

Esto es solucionado de la siguiente manera: Primero se debe tener conocimiento del alfabeto permitido para la clase, el cual está contenido en un arreglo. Luego se escoge un número aleatorio para que represente un índice del arreglo, y finalmente se asigna a la clase el contenido del arreglo que se encuentra en el índice anteriormente seleccionado. Esto sería como sigue:

$$\text{Valor} = \text{alfabeto} [\text{Int}(\text{Rand}() * (\text{Longitud del alfabeto}() - 1))] ,$$

donde $\text{Rand}()$: es una función que devuelve valores entre 0 y 1.

Este es un proceso muy lento, pero afortunadamente, la mutación se da raramente. Igual que en la naturaleza, la mutación es absolutamente rara. Además la mutación permitirá que aparezcan

nuevas características inexistentes en generaciones anteriores. Como es una alteración aleatoria, no es posible garantizar que sea buena. En caso que sea positivo permanecerá en las siguientes generaciones, caso contrario, será eliminada por el proceso de selección.

Para esto se debe definir un porcentaje de ocurrencia del operador de mutación, que nos permita llegar a encontrar una solución factible, es por esta razón que se han realizado un conjunto de pruebas cuyo resultado determina que el mejor es el de 1%. Dichas pruebas se encuentran detalladas en el punto 6.3.2.2.1 del capítulo 6.

4.3.10.Reinserción y Elitismo

Cada vez que se aplican los operadores genéticos tenemos como resultado un número de nuevos individuos (los hijos), los cuales se deben integrar de alguna manera a la población para formar la siguiente generación. Esta operación de reinserción es mejor realizarla tomando en cuenta a los padres y a los hijos que son generados a partir de los primeros, ya que así podemos tener un mayor espacio de búsqueda de los mejores individuos, lo cual no pasaría si solamente consideramos a los individuos hijos. Para demostrar esto se realizaron las respectivas pruebas que están descritas en el punto 6.3.3.1 del capítulo 6.

Otro concepto usado para la reinserción de cromosomas de la población en la siguiente generación, es el elitismo. Este consiste en conservar un número X de mejores individuos ó quitar un número X de malos individuos. Realmente, quitar a los malos individuos es igual que retener los individuos $N-X$ mejores; por lo tanto, ambas son variaciones del elitismo.

El concepto de elitismo es asegurarse de no tener un retroceso evolutivo. El elitismo es una idea aceptable, y de hecho, los sistemas pequeños que requieren un número relativamente pequeño de iteraciones, se comportan bien con elitismo. Sin embargo, cuando la complejidad del sistema crece al igual que el número de iteraciones, el elitismo da una preferencia indebida hacia los mejores individuos, tal que obstaculiza a la exploración de nuevas soluciones. Esto podía conducir a una convergencia peligrosa ó a una convergencia cercana a puntos no óptimos. Los experimentos también lo demuestran y se encuentran descritos en el punto 6.3.3.2 del capítulo 6.

Capítulo 5: Diseño de la Solución

En este capítulo se describirá el diseño de la solución en detalle desde el modelado del negocio, la vista de casos de uso, vista lógica, vista de componentes y vista de despliegue. Incluyéndose además diagramas de actividades, las especificaciones de los casos de uso, diagramas de secuencia, diagrama de clases, modelo de datos, la arquitectura del software y los prototipos de la solución.

5.1. Modelado del Negocio

El modelo del negocio nos permite entender cómo funciona el negocio que se desea automatizar para garantizar que el software desarrollado va a cumplir con su propósito; es por esto, que se debe hacer un estudio en el dominio del negocio para determinar los elementos que intervienen en los procesos del mismo.

Con esa finalidad debemos describir el procedimiento que se sigue para la elaboración de los horarios académicos semestrales en la facultad. Este proceso se realiza en cada uno de los ciclos académicos semestrales, habiendo tres ciclos por cada año académico. Un punto importante para el proceso de elaboración de horarios es la recopilación de la información de los cursos a dictar en el semestre,

la plana de docentes y las aulas disponibles para el periodo a considerar; que son los recursos principales para el cumplimiento de este proceso.

En la facultad hay actualmente dos Escuelas Académicas Profesionales, la de Ingeniería de Sistemas y la de Ingeniería de Software, esta última de reciente creación. Cada Escuela define un Plan de Estudios el cual engloba los cursos de cada especialidad e indirectamente definen el perfil del egresado de cada una de ellas. Los planes de estudios deben ser aprobados por el Consejo de Facultad y son sometidos a revisión cada cierta cantidad de años. Esto explica la existencia actualmente en la facultad de dos Planes de Estudios el de 1997 y el establecido para el 2002. El presente trabajo se refiere a la elaboración de horarios para el Plan de Estudios 2002.

Un plan de estudios consta de cursos que pertenecen a semestres académicos, que van del primero al décimo, y además incluye los cursos electivos. En cada ciclo académico se puede tener uno ó más planes académicos activos.

Con respecto a los docentes, éstos pueden dictar un conjunto de cursos de acuerdo a su ámbito de enseñanza y a su preferencia, y a su vez poseen una disponibilidad horaria que puede variar en cada ciclo académico. Esta información es recopilada por el Departamento Académico, quien unos meses antes del inicio de cada ciclo académico se encarga de enviar un formato para que pueda ser llenado por cada uno de los docentes. Éstos deben cumplir una cierta cantidad de horas de enseñanza de acuerdo a la categoría a la que pertenecen; por ello el número de horas disponibles debe oscilar en un determinado rango para una adecuada asignación de docentes a las clases.

Con respecto a las aulas, éstas poseen un horario disponible total en la semana de lunes a sábado; a excepción de algunas aulas que sólo se encuentran disponibles hasta una determinada hora, ya que son utilizadas por la Unidad de Post Grado.

A partir de la información de los cursos, el siguiente paso es determinar la cantidad de grupos por curso que se deben aperturar en el ciclo académico actual; esto es calculado en base a la demanda de alumnos que se van a matricular en dichos cursos. Para determinar la cantidad de grupos a ser aperturados, se considera la información histórica de los ciclos académicos de años anteriores y la información que se obtiene del Sistema Único de Matrícula donde se indica la cantidad de

alumnos que aprueban o desaprueban cada uno de los cursos, y que incluye también información sobre los que son pre-requisitos para los cursos considerados en el semestre actual.

Al obtenerse toda la información necesaria, la Dirección de Escuela puede iniciar el proceso de elaboración de horarios académicos, el cual puede demorar de tres a cuatro días y es realizada por un número de dos personas.

Una vez detallado cada uno de los procedimientos que se debe seguir para elaborar los horarios académicos, podemos determinar los procesos del negocio y sus respectivos trabajadores de negocio. Existen varios procesos tales como: La recopilación de los datos de los docentes es realizada por el trabajador del negocio, en este caso, la secretaria del Departamento Académico. El establecimiento de la disponibilidad de horarios para cada una de las aulas es determinada por la Dirección de Escuela en coordinación con la Unidad de Post Grado. La proyección de grupos por curso que se deben aperturar en el ciclo académico actual es realizada también por el Director de Escuela.

Todos estos procesos son establecidos para la adecuada asignación de docentes, horarios y aulas a cada una de las clases del ciclo actual. Los horarios académicos semestrales son finalmente elaborados por el Director de Escuela y luego son enviados al Departamento Académico para verificar las asignaciones de horas de dictado de cada uno de los docentes; y de esta manera saber si los docentes cumplen con sus horas de enseñanza de acuerdo a la categoría a la que pertenecen. También son enviados a la Dirección de Escuela para dar su respectiva conformidad. Una vez revisada y aprobada esta información es comunicada al Director de Escuela para que pueda ser transmitida al Consejo de Facultad quien se encarga de aprobarlo. Esta aprobación es comunicada a la Dirección de Escuela para finalmente publicar dichos horarios y de esta manera el alumnado pueda dar inicio a su proceso de matrícula.

Es preciso señalar que el proceso de proyección de grupos aperturados para cada uno de los cursos no será considerado debido a que el alcance del presente trabajo está referido principalmente a la asignación de docentes, horarios y aulas a cada una de las clases del ciclo académico actual.

En la figura que se muestra a continuación podemos observar los procesos del negocio:

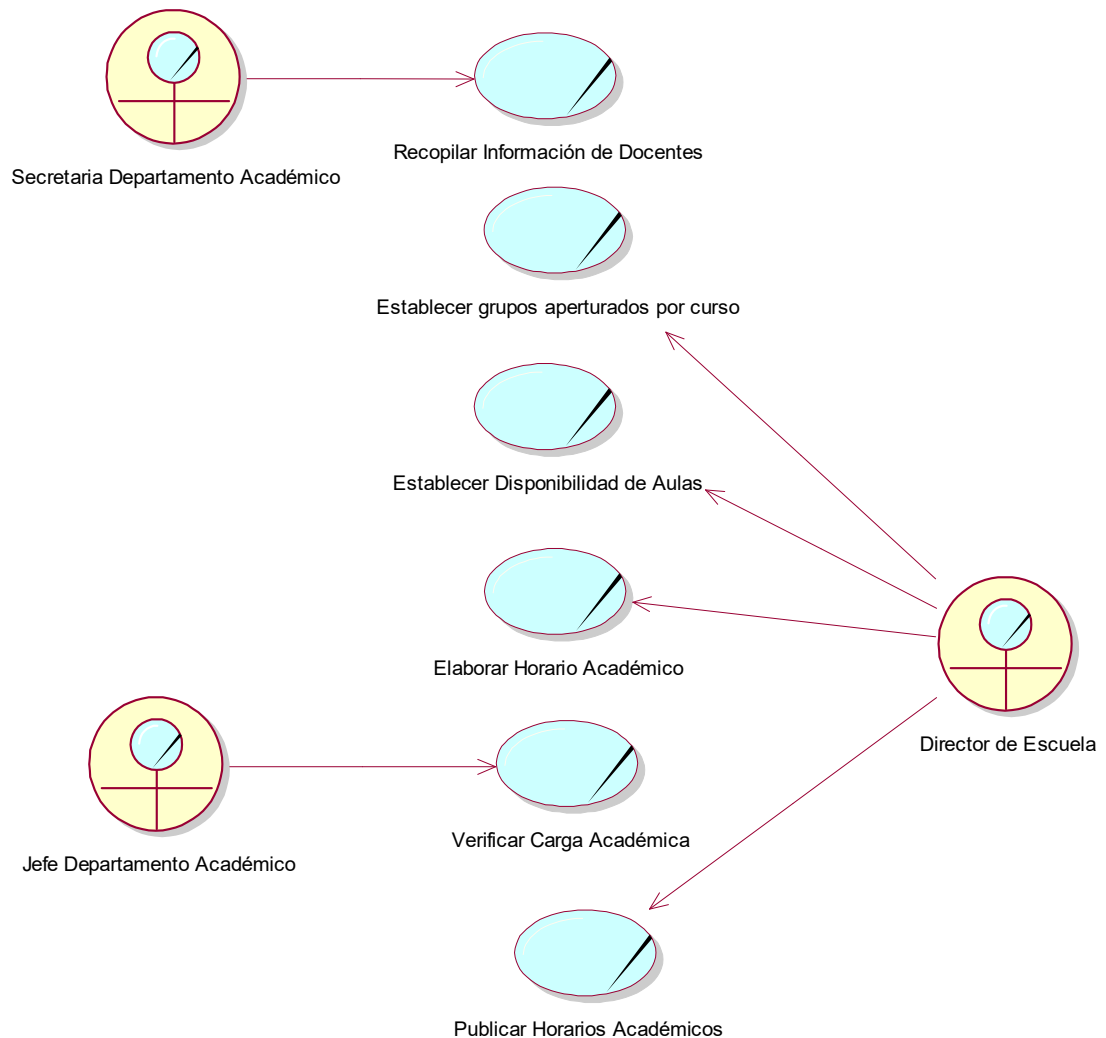


Figura 5.1. Diagrama de Casos de Uso del Negocio

5.1.1. Diagramas de Actividades

Para expresar en orden las tareas que se realizan en el proceso de generación de horarios académicos, vamos a hacer uso de los diagramas de actividades; de esta manera podremos comprender el flujo de trabajo para un posterior análisis.

A continuación presentamos los diagramas de actividades de cada uno de los casos de uso del negocio mostrados anteriormente.

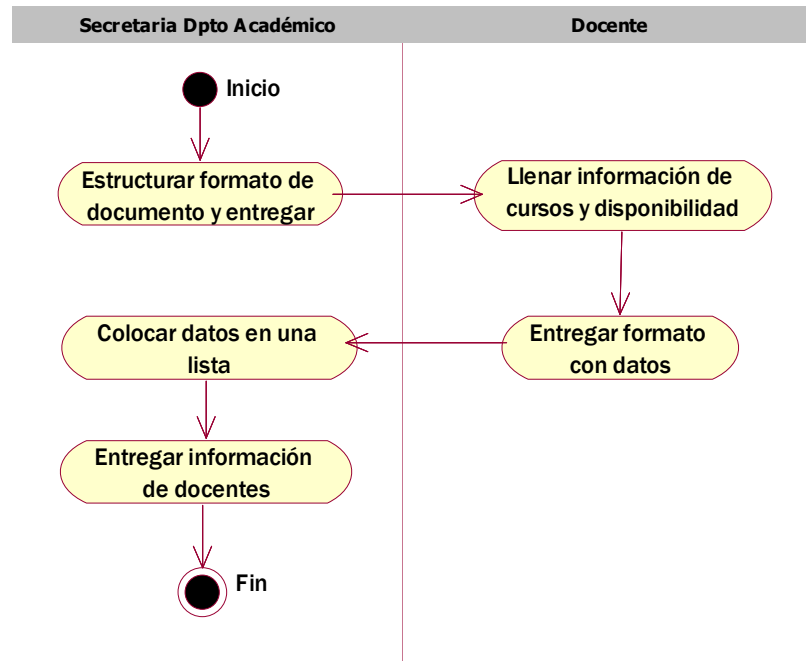


Figura 5.2. Diagrama de Actividad de Recopilar Información de Docentes

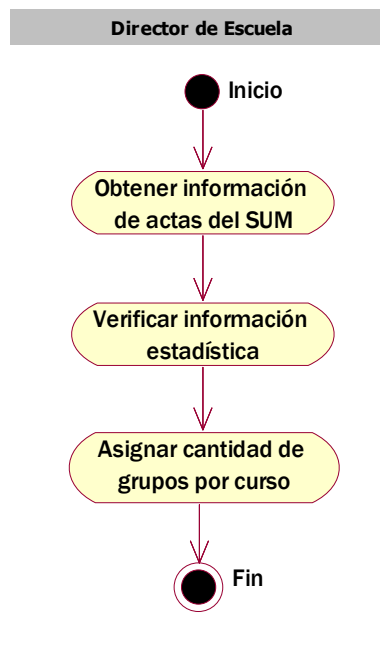


Figura 5.3. Diagrama de Actividad de Establecer grupos aperturados por curso

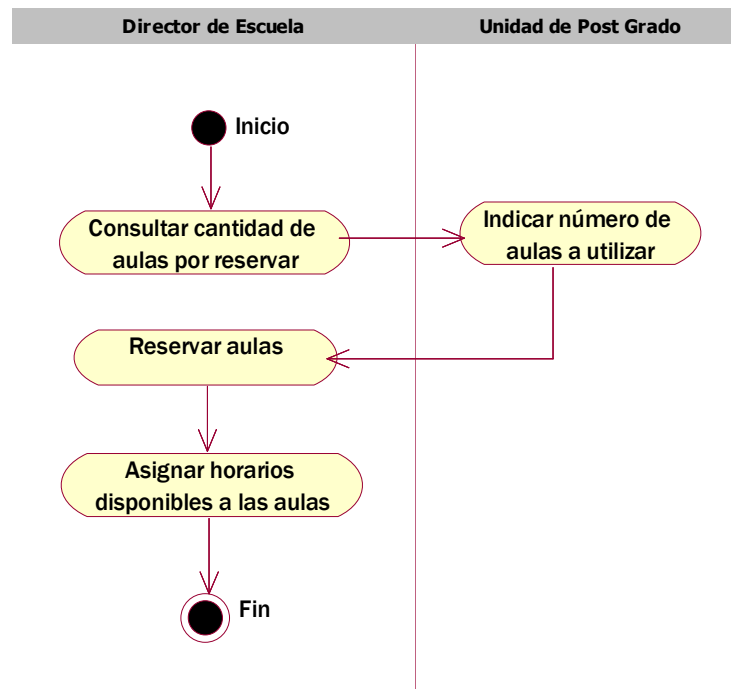


Figura 5.4. Diagrama de Actividad de Establecer Disponibilidad de Aulas

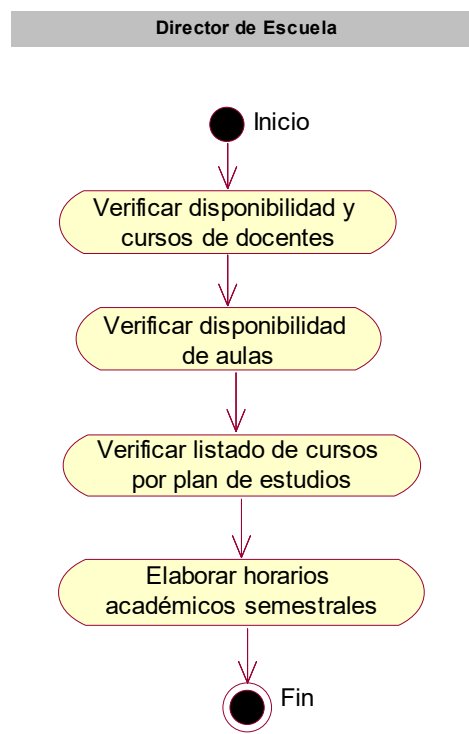


Figura 5.5. Diagrama de Actividad de Elaborar Horario Académico

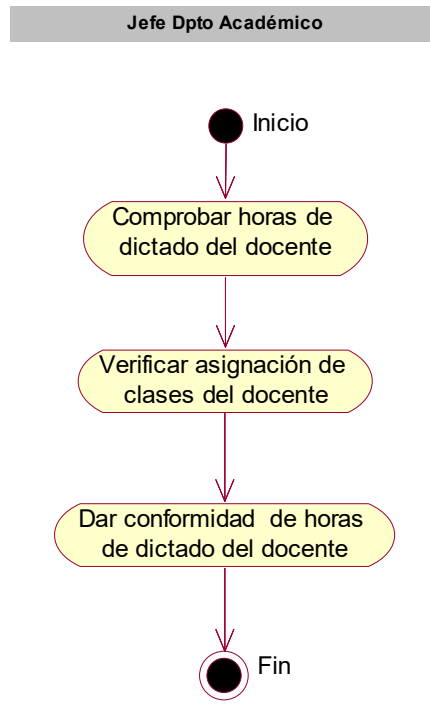


Figura 5.6. Diagrama de Actividad de Verificar Carga Académica

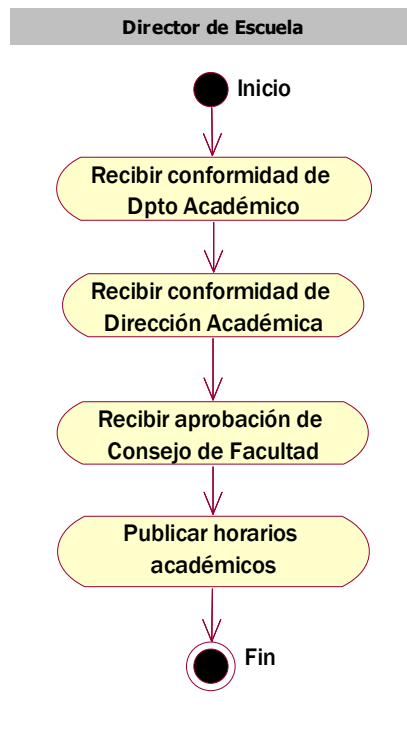


Figura 5.7. Diagrama de Actividad de Publicar Horarios Académicos

5.2. Vista de Casos de Uso

El análisis se realiza con el fin de obtener una visión inicial de las necesidades que deberán ser satisfechas; las cuales nos servirán de base para el diseño.

5.2.1. Diagrama de Casos de Uso

El diagrama de casos de uso que mostramos a continuación permite establecer los distintos requisitos funcionales que se esperan de la aplicación que se va a desarrollar y la forma cómo se relaciona con su entorno.

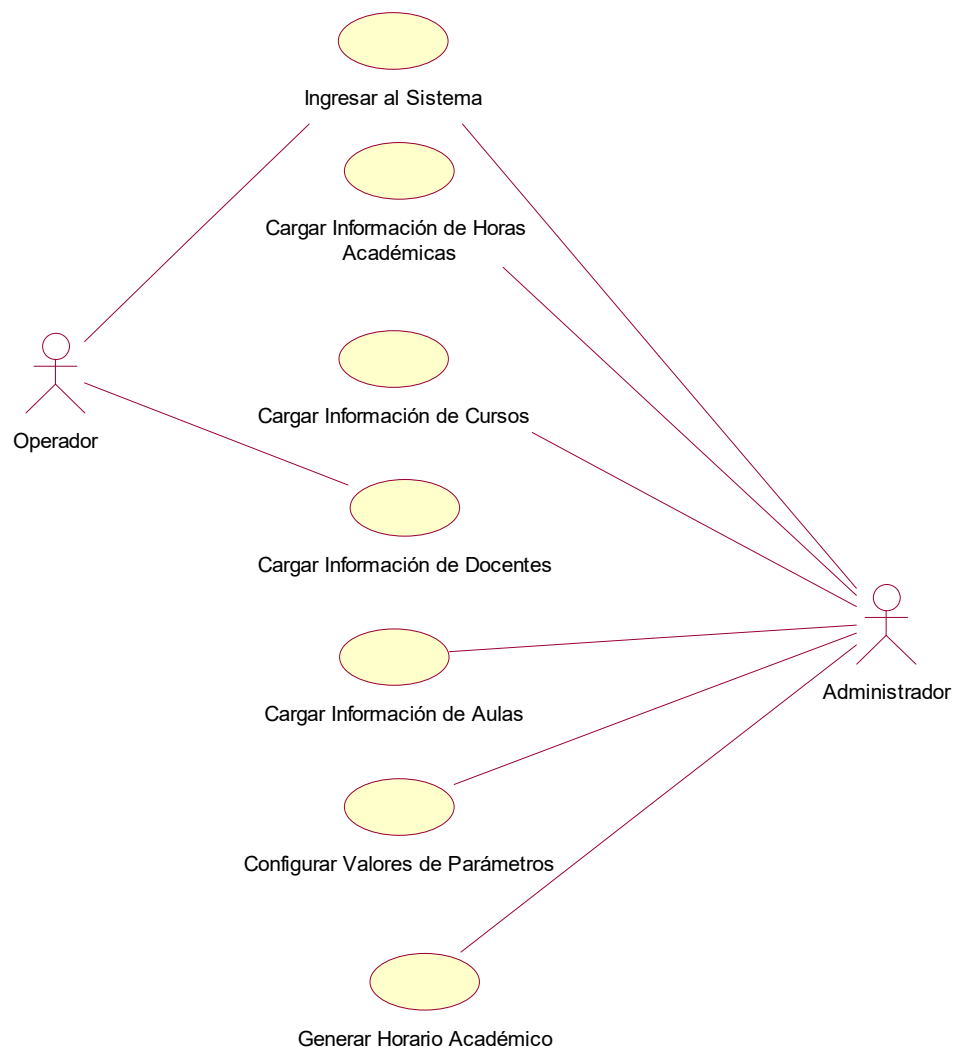


Figura 5.8. Diagrama de Casos de Uso del Sistema

El diagrama muestra un conjunto de casos de uso, actores y sus relaciones. Cada caso de uso denota un requisito funcional que debe presentar el sistema, cuya descripción explica el comportamiento del sistema al afrontar una tarea de negocio; un actor representa un rol que alguien ó algo podría desempeñar, y por último la relación que existe entre un actor y un caso de uso denota la participación de dicho actor en el respectivo caso de uso.

5.2.2. Especificaciones de Casos de Uso y Diagramas de Secuencia

Las especificaciones de casos de uso nos permitirán expresar las necesidades de los usuarios del sistema por ser mecanismos eficientes de comunicación. En seguida pasaremos a detallar los pasos necesarios para cumplir con cada caso de uso; a su vez mostraremos los diagramas de secuencia que nos ayudará a mostrar los objetos que se encuentran en la especificación y la secuencia de mensajes intercambiados entre los objetos para llevar a cabo la funcionalidad descrita.

5.2.2.1. Caso de Uso “Ingresar al Sistema”

1. INGRESAR AL SISTEMA

1.1. Descripción

El caso de uso comienza cuando el Administrador decide ingresar al sistema a través de un usuario y contraseña que son debidamente validados por el sistema. El caso de uso termina cuando se ha permitido al Administrador poder ingresar al sistema.

1.2. Actor (es)

Administrador, Operador.

1.3. Caso de Uso Asociado

No aplica.

2. PRECONDICIONES

No aplica.

3. FLUJO DE EVENTOS

3.1. Flujo básico

- 3.1.1. El Administrador selecciona ingresar al sistema.
- 3.1.2. El sistema muestra la pantalla "Identificación del usuario", solicitando un nombre de usuario y una contraseña.
- 3.1.3. El Administrador ingresa su usuario y contraseña.
- 3.1.4. El Administrador selecciona la opción "Aceptar". Ver flujo alternativo 3.3.1.
- 3.1.5. El sistema valida el usuario y la contraseña ingresados. Ver flujo alternativo 3.3.2.
- 3.1.6. El sistema muestra la pantalla "Menú Principal".
- 3.1.7. El caso de uso termina.

3.2. Sub Flujos

No aplica.

3.3. Flujos alternativos

- 3.3.1. En el paso 3.1.4 del flujo básico.
Si el Administrador selecciona la opción "Cancelar".
 1. El sistema cierra la pantalla "Identificación del usuario".
 2. El caso de uso termina.
- 3.3.2. En el paso 3.1.5 del flujo básico.
Si el Administrador no ingresa un usuario.
 1. El sistema detecta que no ha ingresado el usuario y muestra un mensaje de validación.
 2. El Administrador selecciona la opción "Aceptar".
 3. El sistema cierra el mensaje.
 4. Continúa en el paso 3.1.2 del flujo básico.
Si el Administrador ingresa un usuario no identificado.
 1. El sistema valida que el usuario ingresado es desconocido y muestra un mensaje de validación.
 2. El Administrador selecciona la opción "Aceptar".
 3. Continúa en el paso 3.1.2 del flujo básico.

Si el Administrador no ingresa una contraseña.

1. El sistema detecta que no ha ingresado una contraseña y muestra un mensaje de validación.
2. El Administrador selecciona la opción “Aceptar”.
3. El sistema cierra el mensaje.
4. Continúa en el paso 3.1.2 del flujo básico.

Si el Administrador ingresa una contraseña incorrecta.

1. El sistema valida que la contraseña ingresada es incorrecta y muestra un mensaje de validación.
2. El Administrador selecciona la opción “Aceptar”.
3. Continúa en el paso 3.1.2 del flujo básico.

4. POSCONDICIONES

El Administrador ingresa correctamente al módulo de Generación de Horarios Académicos Semestrales.

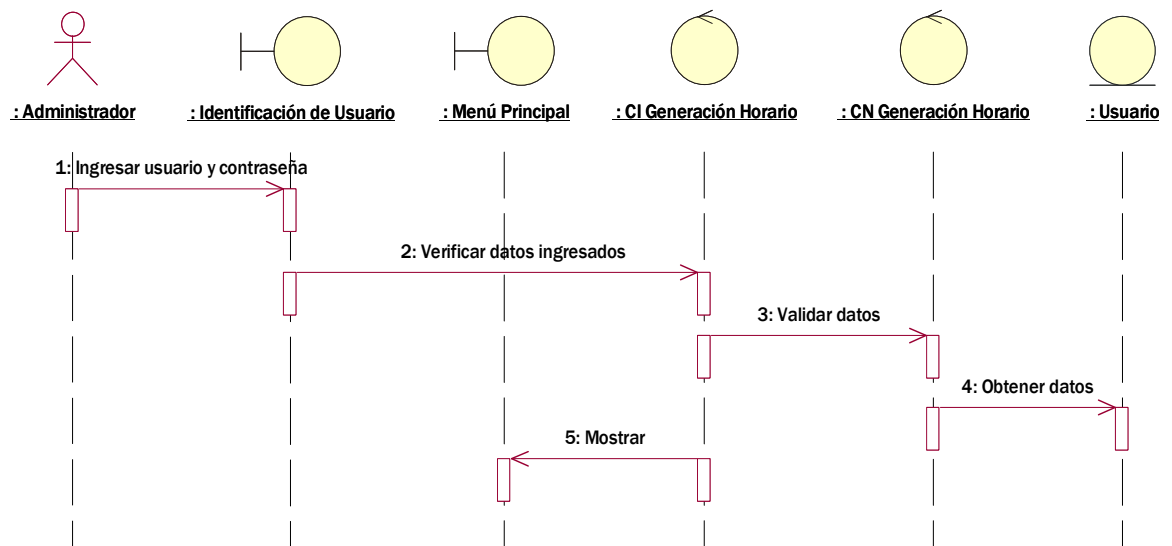


Figura 5.9. Diagrama de Secuencia de "Ingresar al Sistema"

5.2.2.2. Caso de Uso “Cargar Información de Horas Académicas”

1. CARGAR INFORMACIÓN DE HORAS ACADÉMICAS

1.1. Descripción

El caso de uso comienza cuando el Administrador decide ingresar la información de las horas académicas en que los que se puede realizar el dictado de las clases. El proceso empieza cuando se seleccionan los horarios para el dictado de los cursos durante la semana. El caso de uso termina cuando se guarda la información correspondiente a los horarios.

1.2. Actor (es)

Administrador.

1.3. Caso de Uso Asociado

No aplica.

2. PRECONDICIONES

No aplica.

3. FLUJO DE EVENTOS

3.1. Flujo básico

- 3.1.1. El Administrador selecciona la opción “Cargar Información” del menú “Procesos” y luego la opción “Horarios”. Ver flujo alternativo 3.3.1.
- 3.1.2. El sistema muestra la pantalla “Cargar Información de Horarios”.
- 3.1.3. El sistema muestra los horarios seleccionados correspondientes a todos los días de la semana.
- 3.1.4. El Administrador quita la selección de los horarios en los cuales no se puede realizar el dictado de clases.
- 3.1.5. El sistema muestra marcados los horarios seleccionados.
- 3.1.6. El Administrador selecciona la opción “Guardar”. Ver flujo alternativo 3.3.2.

- 3.1.7. El sistema graba la información asociada a los horarios y muestra un mensaje de información.
- 3.1.8. El Administrador acepta el mensaje mostrado.
- 3.1.9. El Administrador selecciona la opción “Salir”.
- 3.1.10. El sistema cierra la pantalla “Cargar Información de Horarios”.
- 3.1.11. El caso de uso termina.

3.2. Sub Flujos

No aplica.

3.3. Flujos alternativos

- 3.3.1. En el paso 3.1.1 del flujo básico.
 - Si el Administrador verifica que el ciclo académico mostrado no corresponde al actual.
 - 1. El Administrador selecciona el criterio correspondiente a ciclo académico.
 - 2. El sistema muestra la lista de ciclos académicos.
 - 3. El Administrador selecciona el ciclo académico actual.
 - 4. El sistema muestra el ciclo académico seleccionado.
 - 5. Continúa en el paso 3.1.1 del flujo básico.
- 3.3.2. En el paso 3.1.6 del flujo básico.
 - Si el Administrador selecciona la opción “Cancelar”.
 - 1. Continúa en el paso 3.1.3 del flujo básico.
 - Si el Administrador selecciona la opción “Salir”.
 - 1. Continúa en el paso 3.1.10 del flujo básico.
 - Si el Administrador decide seleccionar los horarios desmarcados.
 - 1. El Administrador selecciona los horarios en los cuales se puede realizar el dictado de clases.
 - 2. El sistema muestra marcados los horarios seleccionados.
 - 3. Continúa en el paso 3.1.6 del flujo básico.

4. POSCONDICIONES

La Información de los horarios de la semana, en que se realiza el dictado de clases, está cargada en el sistema.

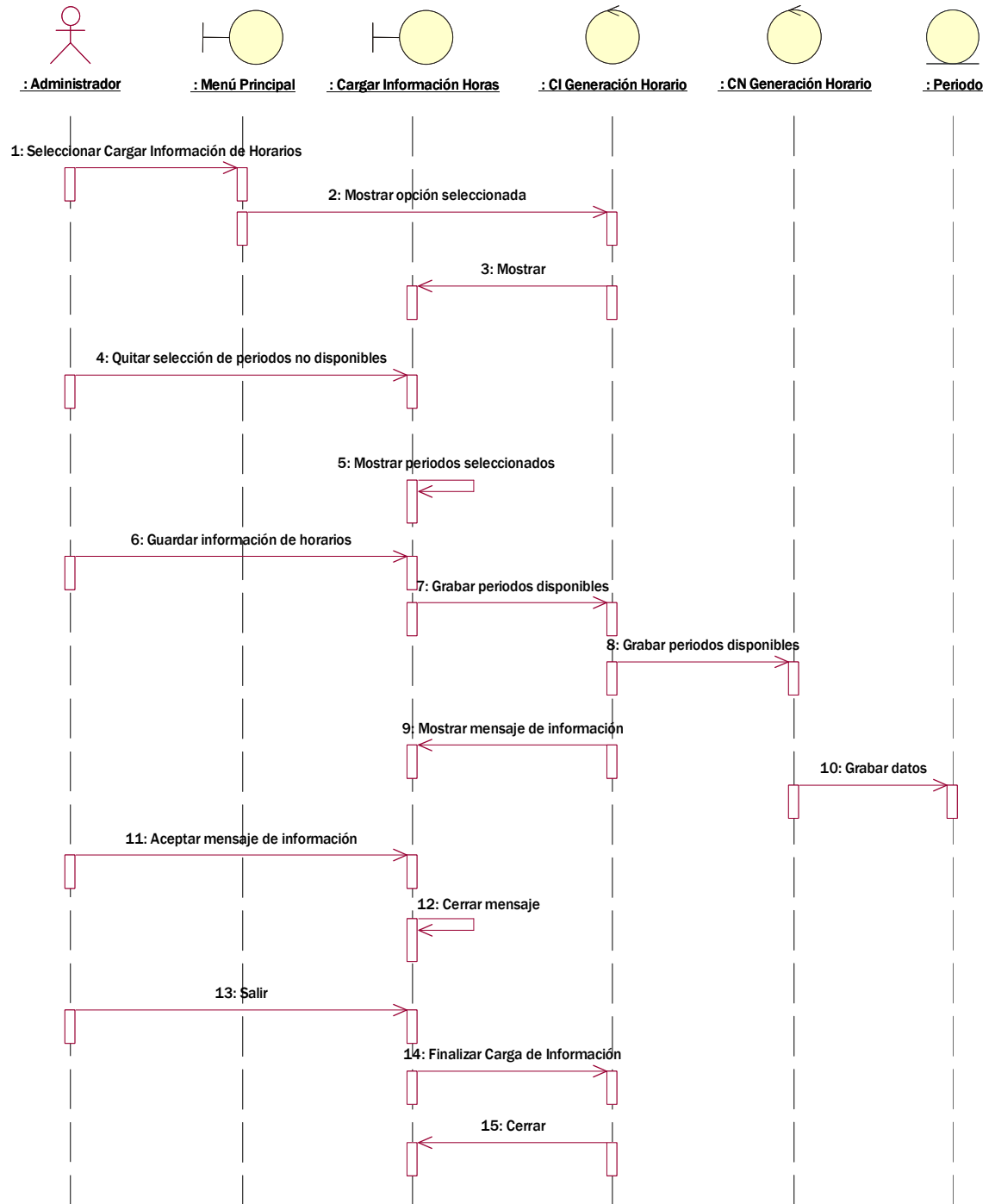


Figura 5.10. Diagrama de Secuencia de “Cargar Información de Horas Académicas”

5.2.2.3. Caso de Uso “Cargar Información de Cursos”

1. CARGAR INFORMACIÓN DE CURSOS

1.1. Descripción

El caso de uso comienza cuando el Administrador decide cargar la información de los cursos. El proceso empieza cuando se carga la información desde un archivo en formato excel a la base de datos, y al término del proceso se notifica al usuario que la carga fue satisfactoria. Además debe ingresar los tipos de dictado asociados a cada curso, así como también las horas de dictado para cada uno de los tipos de cursos. El caso de uso termina cuando se guarda la información correspondiente a cada uno de los cursos.

1.2. Actor (es)

Administrador.

1.3. Caso de Uso Asociado

No aplica.

2. PRECONDICIONES

No aplica.

3. FLUJO DE EVENTOS

3.1. Flujo básico

- 3.1.1. El Administrador selecciona la opción “Cargar Información” del menú “Procesos” y luego la opción “Cursos”. Ver flujo alternativo 3.3.1.
- 3.1.2. El sistema muestra la pantalla “Cargar Información de Cursos”.
- 3.1.3. El Administrador selecciona el plan de estudios actual.
- 3.1.4. El sistema muestra el plan de estudios seleccionado.
- 3.1.5. El Administrador selecciona la opción “Buscar”.
- 3.1.6. El sistema muestra un listado de directorios.
- 3.1.7. El Administrador selecciona el archivo que contiene el listado de cursos.

- 3.1.8. El sistema muestra la ruta completa de la ubicación del archivo de cursos.
- 3.1.9. El Administrador selecciona la opción “Cargar”.
- 3.1.10. El sistema graba la información del listado de cursos y muestra un mensaje de información.
- 3.1.11. El Administrador acepta el mensaje mostrado.
- 3.1.12. El sistema muestra la lista de cursos.
- 3.1.13. El Administrador selecciona cada curso para establecer los tipos y las horas de dictado correspondiente a cada uno de ellos.
- 3.1.14. El sistema muestra el nombre del curso, el semestre al que pertenece, la cantidad de grupos aperturados y la cantidad posible de alumnos a matricularse; así como los datos parametrizables correspondientes al curso.
- 3.1.15. El Administrador selecciona la opción “Crear”.
- 3.1.16. El sistema habilita los datos parametrizables correspondientes al curso.
- 3.1.17. El Administrador selecciona un determinado tipo de clasificación e ingresa la cantidad de horas de dictado asociada.
- 3.1.18. El Administrador selecciona el turno en que puede ser dictado el curso. Ver flujo alternativo 3.3.2.
- 3.1.19. El sistema muestra los datos ingresados.
- 3.1.20. El Administrador selecciona la opción “Guardar”. Ver flujo alternativo 3.3.3.
- 3.1.21. El sistema graba la información asociada al curso y muestra un mensaje de información.
- 3.1.22. El Administrador acepta el mensaje mostrado.
- 3.1.23. El Administrador selecciona la opción “Salir”.
- 3.1.24. El sistema cierra la pantalla “Cargar Información de Cursos”.
- 3.1.25. El caso de uso termina.

3.2. Sub Flujos

3.2.1. Modificar datos del curso

Si el Administrador decide modificar los valores de los datos parametrizables de un curso.

1. El Administrador selecciona un determinado curso.
2. El sistema muestra los valores de los datos parametrizables del curso seleccionado y habilita la opción "Guardar".
3. El Administrador realiza las modificaciones respectivas.
4. El Administrador selecciona la opción "Guardar". Ver flujo alternativo 3.3.3.
5. El sistema graba la información asociada al curso y muestra un mensaje de información.
6. El Administrador acepta el mensaje mostrado.
7. Continúa en el paso 3.1.23 del flujo básico.

3.2.2. Eliminar datos del curso

Si el Administrador decide eliminar los valores de los datos parametrizables de un curso.

1. El Administrador selecciona un determinado curso.
2. El sistema muestra los valores de los datos parametrizables del curso seleccionado y habilita la opción "Guardar".
3. El Administrador selecciona la opción "Eliminar".
4. El sistema muestra un mensaje de confirmación.
5. El Administrador confirma el mensaje mostrado. Ver flujo alternativo 3.3.4.
6. El sistema elimina los valores asociados al curso y muestra un mensaje de información.
7. El Administrador acepta el mensaje mostrado.
8. Continúa en el paso 3.1.23 del flujo básico.

3.3. Flujos alternativos

3.3.1. En el paso 3.1.1 del flujo básico.

Si el Administrador verifica que el ciclo académico mostrado no corresponde al actual.

1. El Administrador selecciona el criterio correspondiente a ciclo académico.
2. El sistema muestra la lista de ciclos académicos.
3. El Administrador selecciona el ciclo académico actual.
4. El sistema muestra el ciclo académico seleccionado.
5. Continúa en el paso 3.1.1 del flujo básico.

3.3.2. En el paso 3.1.18 del flujo básico.

Si el Administrador decide realizar una asignación previa de un horario al curso.

1. El Administrador selecciona el horario específico que va ser asignado al curso.
2. El sistema muestra marcados los horarios seleccionados.
3. Continúa en el paso 3.1.20 del flujo básico.

3.3.3. En el paso 3.1.20 del flujo básico y en el paso 3.2.1-4 del sub flujo “Modificar datos del curso”.

Si el Administrador selecciona la opción “Cancelar”.

1. Continúa en el paso 3.1.12 del flujo básico.

3.3.4. En el paso 3.2.2-5 del sub flujo “Eliminar datos del curso”.

Si el Administrador no confirma el mensaje para eliminar los datos asociados al curso.

1. El sistema cierra el mensaje.
2. Continúa en el paso 3.1.12 del flujo básico.

4. POSCONDICIONES

La Información de los cursos está cargada en el sistema.

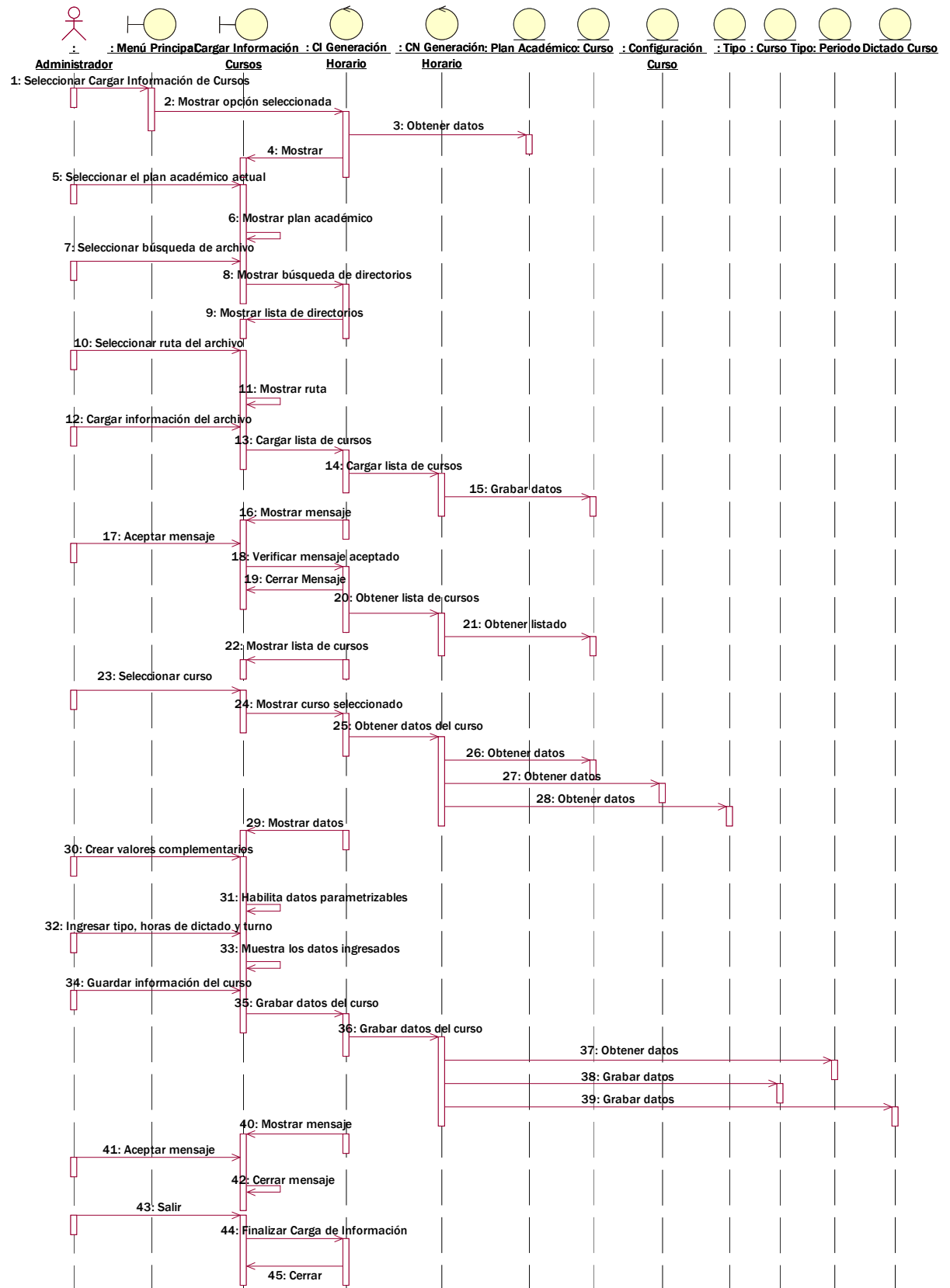


Figura 5.11. Diagrama de Secuencia de “Cargar Información de Cursos”

5.2.2.4. Caso de Uso “Cargar Información de Docentes”

1. CARGAR INFORMACIÓN DE DOCENTES

1.1. Descripción

El caso de uso comienza cuando el Operador decide cargar la información de los docentes. El proceso empieza cuando se carga la información desde un archivo en formato excel a la base de datos, y al término del proceso se notifica al usuario que la carga fue satisfactoria. Además debe ingresar la disponibilidad de horario y los cursos de preferencia de cada docente. El caso de uso termina cuando se guarda la información correspondiente a cada uno de los docentes.

1.2. Actor (es)

Administrador.

1.3. Caso de Uso Asociado

No aplica.

2. PRECONDICIONES

La Información correspondiente a los cursos debe existir en el sistema, al igual que la información de horarios en los que se puede realizar el dictado de clases.

3. FLUJO DE EVENTOS

3.1. Flujo básico

- 3.1.1. El Administrador selecciona la opción “Cargar Información” del menú “Procesos” y luego la opción “Docentes”. Ver flujo alternativo 3.3.1.
- 3.1.2. El sistema muestra la pantalla “Cargar Información de Docentes”.
- 3.1.3. El Administrador selecciona la opción “Buscar”.
- 3.1.4. El sistema muestra un listado de directorios.
- 3.1.5. El Administrador selecciona el archivo que contiene el listado de docentes.
- 3.1.6. El sistema muestra la ruta completa de la ubicación del archivo de docentes.

- 3.1.7. El Administrador selecciona la opción “Cargar”.
- 3.1.8. El sistema graba la información del listado de docentes y muestra un mensaje de información.
- 3.1.9. El Administrador acepta el mensaje mostrado.
- 3.1.10. El sistema muestra la lista de docentes.
- 3.1.11. El Administrador selecciona un docente determinado para establecer su disponibilidad horaria y los cursos de su preferencia.
- 3.1.12. El sistema muestra el nombre y la categoría del docente, así como los datos parametrizables correspondientes al docente.
- 3.1.13. El Administrador selecciona los horarios disponibles del docente.
- 3.1.14. El sistema muestra marcados los horarios seleccionados.
- 3.1.15. El Administrador selecciona los cursos de preferencia del docente.
- 3.1.16. El sistema muestra en una lista los cursos seleccionados.
- 3.1.17. El Administrador selecciona la opción “Guardar”.
Ver flujo alternativo 3.3.2.
- 3.1.18. El sistema graba la información asociada al docente y muestra un mensaje de información.
- 3.1.19. El Administrador acepta el mensaje mostrado.
- 3.1.20. El Administrador selecciona la opción “Salir”.
- 3.1.21. El sistema cierra la pantalla “Cargar Información de Docentes”.
- 3.1.22. El caso de uso termina.

3.2. Sub Flujos

3.2.1 Modificar datos del docente

Si el Administrador decide modificar los valores de los datos parametrizables de un docente.

1. El Administrador selecciona un determinado docente.

2. El sistema muestra los valores de los datos parametrizables del docente seleccionado y habilita la opción “Guardar”.
3. El Administrador realiza las modificaciones respectivas.
4. El Administrador selecciona la opción “Guardar”.
Ver flujo alternativo 3.3.3.
5. El sistema graba la información asociada al docente y muestra un mensaje de información.
6. El Administrador acepta el mensaje mostrado.
7. Continúa en el paso 3.1.20 del flujo básico.

3.2.2 Eliminar datos del docente

Si el Administrador decide eliminar los valores de los datos parametrizables de un docente.

1. El Administrador selecciona un determinado docente.
2. El sistema muestra los valores de los datos parametrizables del docente seleccionado y habilita la opción “Guardar”.
3. El Administrador selecciona la opción “Eliminar”.
4. El sistema muestra un mensaje de confirmación.
5. El Administrador confirma el mensaje mostrado.
Ver flujo alternativo 3.3.4.
6. El sistema elimina los valores asociados al docente y muestra un mensaje de información.
7. El Administrador acepta el mensaje mostrado.
8. Continúa en el paso 3.1.20 del flujo básico.

3.3. Flujos alternativos

3.3.1 En el paso 3.1.1 del flujo básico.

Si el Administrador verifica que el ciclo académico mostrado no corresponde al actual.

1. El Administrador selecciona el criterio correspondiente a ciclo académico.
2. El sistema muestra la lista de ciclos académicos.

3. El Administrador selecciona el ciclo académico actual.
4. El sistema muestra el ciclo académico seleccionado.
5. Continúa en el paso 3.1.1 del flujo básico.

3.3.2 En el paso 3.1.17 del flujo básico.

Si el Administrador selecciona la opción “Cancelar”.

1. Continúa en el paso 3.1.10 del flujo básico.

Si el Administrador selecciona la opción “Salir”.

1. Continúa en el paso 3.1.21 del flujo básico.

Si el Administrador decide realizar una asignación previa de un curso al docente.

1. El Administrador selecciona el curso específico que va ser asignado al docente.
2. El sistema muestra en una lista el curso seleccionado.
3. Continúa en el paso 3.1.17 del flujo básico.

3.3.3 En el paso 3.2.1-4 del sub flujo “Modificar datos del docente”.

Si el Administrador selecciona la opción “Cancelar”.

1. Continúa en el paso 3.1.10 del flujo básico.

Si el Administrador selecciona la opción “Salir”.

1. Continúa en el paso 3.1.21 del flujo básico.

3.3.4 En el paso 3.2.2-5 del sub flujo “Eliminar datos del docente”.

Si el Administrador no confirma el mensaje para eliminar los datos asociados al docente.

1. El sistema cierra el mensaje.
2. Continúa en el paso 3.1.10 del flujo básico.

4. POSCONDICIONES

La Información de los docentes está cargada en el sistema.

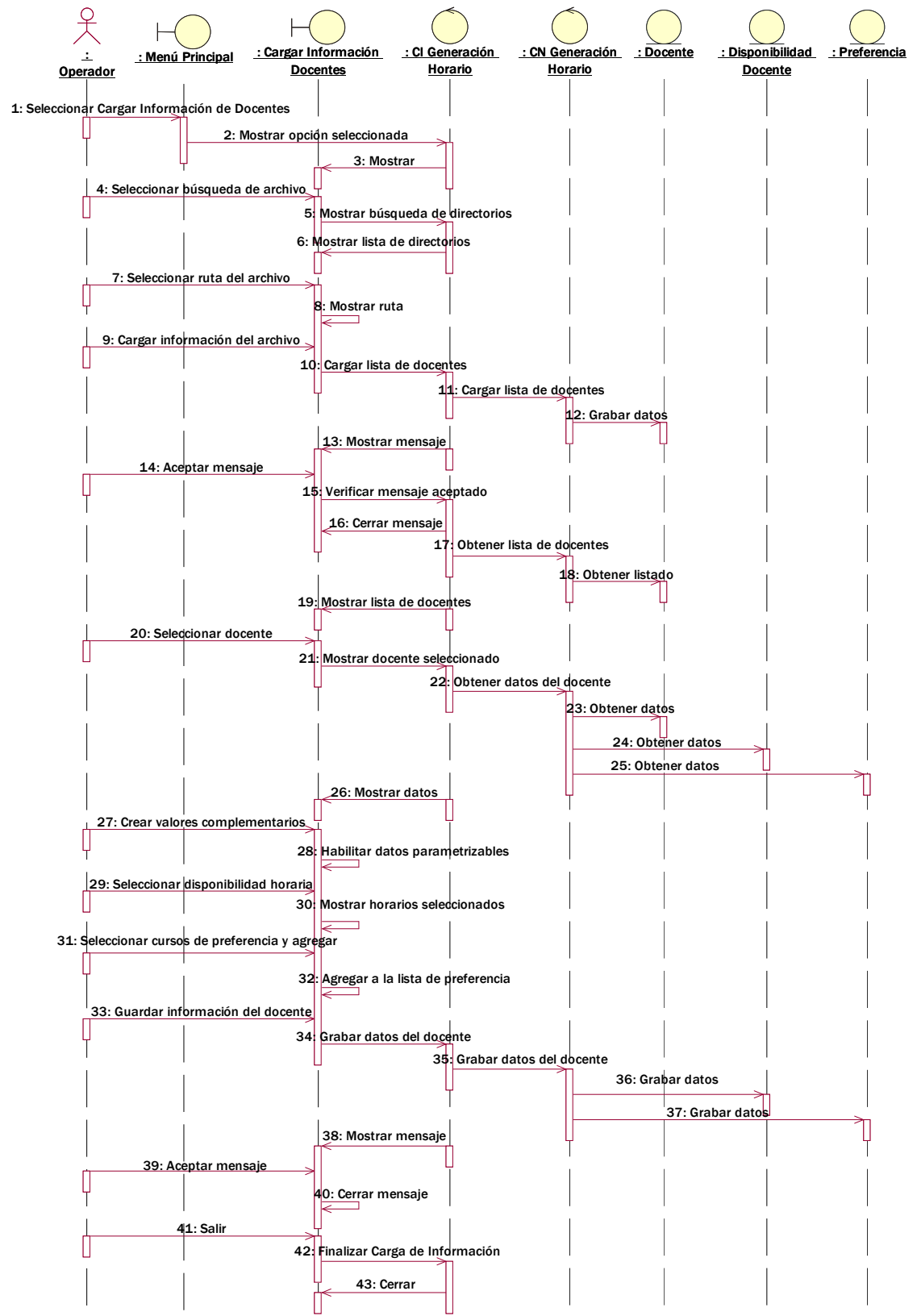


Figura 5.12. Diagrama de Secuencia de “Cargar Información de Docentes”

5.2.2.5. Caso de Uso “Cargar Información de Aulas”

1. CARGAR INFORMACIÓN DE AULAS

1.1. Descripción

El caso de uso comienza cuando el Administrador decide cargar la información de las aulas. El proceso empieza cuando se carga la información desde un archivo en formato excel a la base de datos, y al término del proceso se notifica al usuario que la carga fue satisfactoria. Además debe ingresar los horarios disponibles durante la semana de cada aula. El caso de uso termina cuando se guarda la información correspondiente a cada una de las aulas.

1.2. Actor (es)

Administrador.

1.3. Caso de Uso Asociado

No aplica.

2. PRECONDICIONES

No aplica.

3. FLUJO DE EVENTOS

3.1. Flujo básico

- 3.1.1. El Administrador selecciona la opción “Cargar Información” del menú “Procesos” y luego la opción “Aulas”. Ver flujo alternativo 3.3.1.
- 3.1.2. El sistema muestra la pantalla “Cargar Información de Aulas”.
- 3.1.3. El Administrador selecciona la opción “Buscar”.
- 3.1.4. El sistema muestra un listado de directorios.
- 3.1.5. El Administrador selecciona el archivo que contiene el listado de aulas.
- 3.1.6. El sistema muestra la ruta completa de la ubicación del archivo de aulas.
- 3.1.7. El Administrador selecciona la opción “Cargar”.
- 3.1.8. El sistema graba la información del listado de aulas y muestra un mensaje de información.

- 3.1.9. El Administrador acepta el mensaje mostrado.
- 3.1.10. El sistema muestra la lista de aulas.
- 3.1.11. El Administrador selecciona un aula determinado para establecer su disponibilidad a lo largo de la semana.
- 3.1.12. El sistema muestra el nombre, el tipo y la capacidad del aula, así como los datos parametrizables correspondientes al aula.
- 3.1.13. El sistema muestra seleccionados los horarios hábiles correspondientes a los días de la semana.
- 3.1.14. El Administrador quita la selección de las horas en las cuales el aula no está disponible.
- 3.1.15. El sistema muestra marcados los horarios disponibles del aula.
- 3.1.16. El Administrador selecciona la opción “Guardar”. Ver flujo alternativo 3.3.2.
- 3.1.17. El sistema graba la información asociada al aula y muestra un mensaje de información.
- 3.1.18. El Administrador acepta el mensaje mostrado.
- 3.1.19. El Administrador selecciona la opción “Salir”.
- 3.1.20. El sistema cierra la pantalla “Cargar Información de Aulas”.
- 3.1.21. El caso de uso termina.

3.2. Sub Flujos

3.2.1 Modificar datos del aula

Si el Administrador decide modificar los valores de los datos parametrizables de un aula.

1. El Administrador selecciona una determinada aula.
2. El sistema muestra los valores de los datos parametrizables del aula seleccionada y habilita la opción “Guardar”.
3. El Administrador realiza las modificaciones respectivas.

4. El Administrador selecciona la opción “Guardar”.
Ver flujo alternativo 3.3.2.
5. El sistema graba la información asociada al aula y muestra un mensaje de información.
6. El Administrador acepta el mensaje mostrado.
7. Continúa en el paso 3.1.19 del flujo básico.

3.2.2 Eliminar datos del aula

Si el Administrador decide eliminar los valores de los datos parametrizables de un aula.

1. El Administrador selecciona una determinada aula.
2. El sistema muestra los valores de los datos parametrizables del aula seleccionada y habilita la opción “Guardar”.
3. El Administrador selecciona la opción “Eliminar”.
4. El sistema muestra un mensaje de confirmación.
5. El Administrador confirma el mensaje mostrado.
Ver flujo alternativo 3.3.3.
6. El sistema elimina los valores asociados al docente y muestra un mensaje de información.
7. El Administrador acepta el mensaje mostrado.
8. Continúa en el paso 3.1.19 del flujo básico.

3.3. Flujos alternativos

3.3.1 En el paso 3.1.1 del flujo básico.

Si el Administrador verifica que el ciclo académico mostrado no corresponde al actual.

1. El Administrador selecciona el criterio correspondiente a ciclo académico.
2. El sistema muestra la lista de ciclos académicos.
3. El Administrador selecciona el ciclo académico actual.
4. El sistema muestra el ciclo académico seleccionado.

5. Continúa en el paso 3.1.1 del flujo básico.

3.3.2 En el paso 3.1.16 del flujo básico y en el paso 3.2.1-4 del sub flujo “Modificar datos del aula”.

Si el Administrador selecciona la opción “Cancelar”.

1. Continúa en el paso 3.1.10 del flujo básico.

Si el Administrador selecciona la opción “Salir”.

1. Continúa en el paso 3.1.20 del flujo básico.

Si el Administrador decide seleccionar los horarios hábiles desmarcados.

1. El Administrador selecciona las horas en las cuales el aula está disponible.

2. El sistema muestra marcados los horarios seleccionados.

3. Continúa en el paso 3.1.16 del flujo básico.

3.3.3 En el paso 3.2.2-5 del sub flujo “Eliminar datos del aula”.

Si el Administrador no confirma el mensaje para eliminar los datos asociados al aula.

1. El sistema cierra el mensaje.

2. Continúa en el paso 3.1.10 del flujo básico.

4. POSCONDICIONES

La Información de disponibilidad de las aulas está cargada en el sistema.

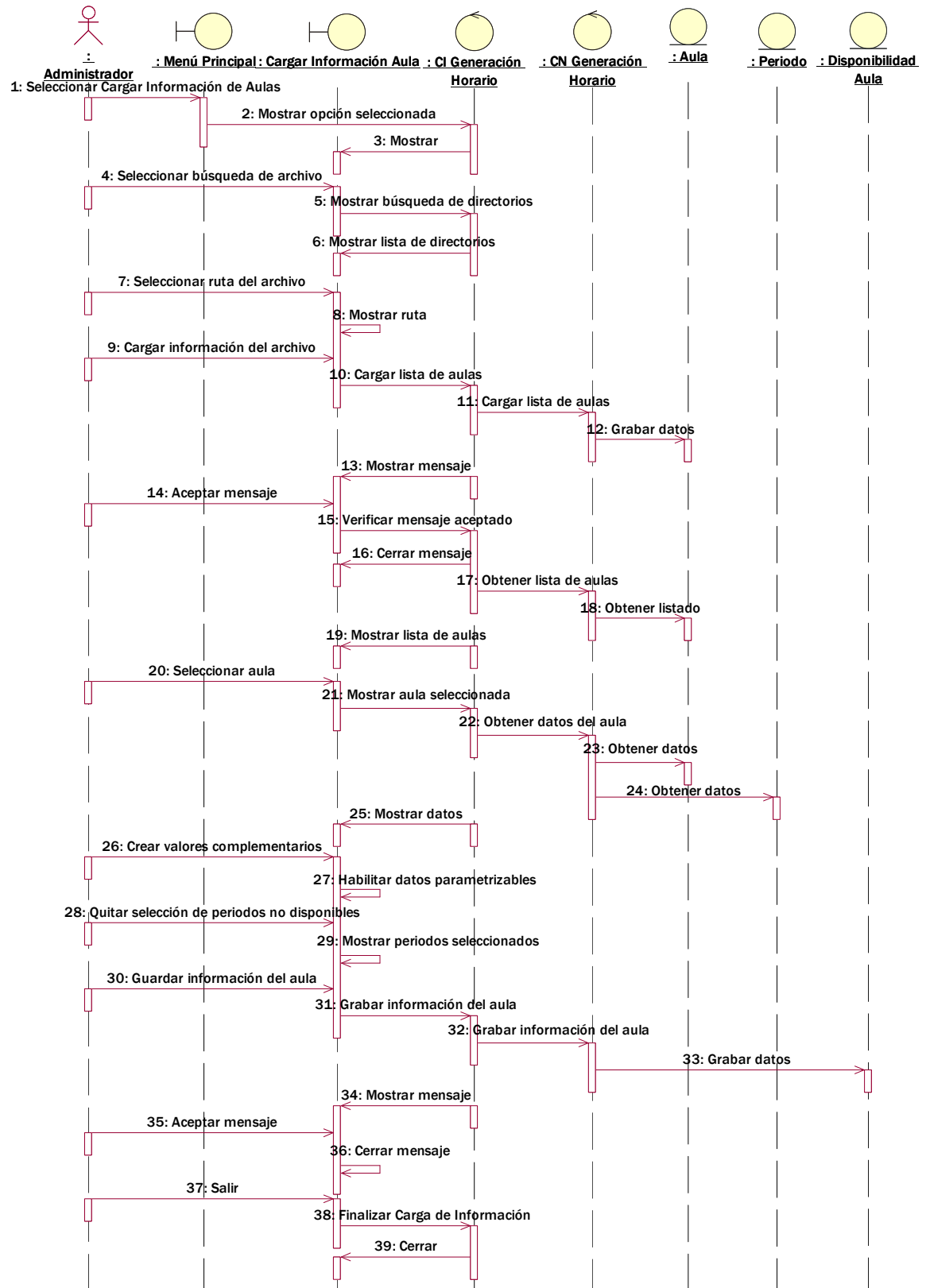


Figura 5.13. Diagrama de Secuencia de “Cargar Información de Aulas”

5.2.2.6. Caso de Uso “Configurar Valores de Parámetros”

1. CONFIGURAR VALORES DE PARÁMETROS

1.1. Descripción

El caso de uso comienza cuando el Administrador decide configurar los valores de los parámetros necesarios para el funcionamiento del algoritmo genético. El proceso empieza cuando se ingresan los valores de cada uno de los parámetros de configuración. El caso de uso termina cuando se guarda la información correspondiente a cada uno de los parámetros.

1.2. Actor (es)

Administrador.

1.3. Caso de Uso Asociado

No aplica.

2. PRECONDICIONES

No aplica.

3. FLUJO DE EVENTOS

3.1. Flujo básico

- 3.1.1. El Administrador selecciona la opción “Configurar Valores de Parámetros” del menú “Procesos”.
- 3.1.2. El sistema muestra la pantalla “Configurar Valores de Parámetros”.
- 3.1.3. El Administrador ingresa los valores para cada parámetro de la configuración, los cuales son el tamaño de la población y el número de generaciones.
- 3.1.4. El sistema muestra los valores ingresados.
- 3.1.5. El Administrador selecciona la opción “Guardar”. Ver flujo alternativo 3.3.1.
- 3.1.6. El sistema valida los valores ingresados. Ver flujo alternativo 3.3.2.
- 3.1.7. El sistema graba la información asociada a los parámetros y muestra un mensaje de información.

- 3.1.8. El Administrador acepta el mensaje mostrado.
- 3.1.9. El Administrador selecciona la opción “Salir”.
- 3.1.10. El sistema cierra la pantalla “Configurar Valores de Parámetros”.
- 3.1.11. El caso de uso termina.

3.2. Sub Flujos

No aplica.

3.3. Flujos alternativos

3.3.1 En el paso 3.1.1 del flujo básico.

Si el Administrador selecciona la opción “Cancelar”.

1. Continúa en el paso 3.1.2 del flujo básico.

Si el Administrador selecciona la opción “Salir”.

1. Continúa en el paso 3.1.10 del flujo básico.

3.3.2 En el paso 3.1.6 del flujo básico.

Si el Administrador no ingresa un tamaño para la población.

1. El sistema detecta que no ha ingresado el tamaño de la población y muestra un mensaje de validación.
2. El Administrador selecciona la opción “Aceptar”.
3. El sistema cierra el mensaje.
4. Continúa en el paso 3.1.3 del flujo básico.

Si el Administrador ingresa un tamaño para la población no válido.

1. El sistema valida que el tamaño de la población es incorrecto y muestra un mensaje de validación.
2. El Administrador selecciona la opción “Aceptar”.
3. Continúa en el paso 3.1.3 del flujo básico.

Si el Administrador no ingresa un número de generaciones.

1. El sistema detecta que no ha ingresado el número de generaciones y muestra un mensaje de validación.

2. El Administrador selecciona la opción “Aceptar”.
3. El sistema cierra el mensaje.
4. Continúa en el paso 3.1.3 del flujo básico.

Si el Administrador ingresa un número de generaciones no válido.

1. El sistema valida que el número de generaciones es incorrecto y muestra un mensaje de validación.
2. El Administrador selecciona la opción “Aceptar”.
3. Continúa en el paso 3.1.3 del flujo básico.

4. POSCONDICIONES

La Información de cada uno de los valores de configuración está cargada en el sistema.

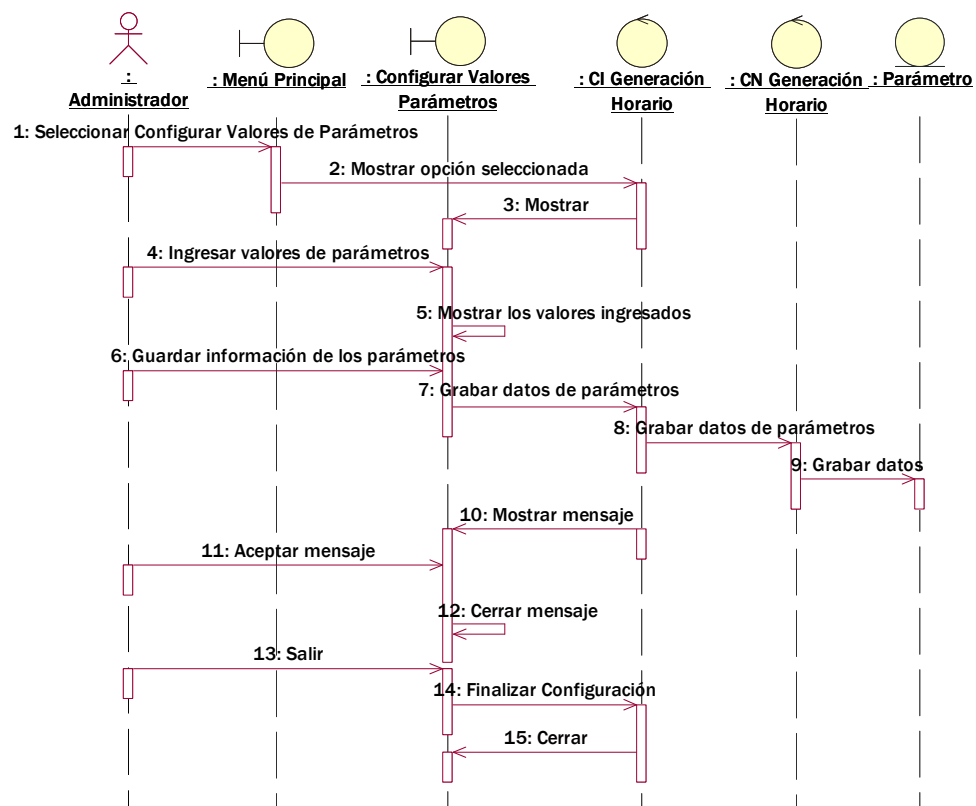


Figura 5.14. Diagrama de Secuencia de “Configurar Valores de Parámetros”

5.2.2.7. Caso de Uso “Generar Horario Académico”

1. GENERAR HORARIO ACADÉMICO

1.1. Descripción

El caso de uso comienza cuando el Administrador decide generar los horarios académicos semestrales. El proceso empieza cuando se ejecuta el algoritmo genético para que se proceda a generar los horarios de los cursos de cada uno de los semestres del ciclo actual. El caso de uso termina cuando se muestra la información correspondiente a los cursos, docentes, horarios y aulas relacionados entre sí que formarán un horario académico para cada uno de los semestres.

1.2. Actor (es)

Administrador.

1.3. Caso de Uso Asociado

No aplica.

2. PRECONDICIONES

No aplica.

3. FLUJO DE EVENTOS

3.1. Flujo básico

- 3.1.1. El Administrador selecciona la opción “Generar Horario Académico” del menú “Procesos”.
- 3.1.2. El sistema muestra la pantalla “Generar Horario Académico”.
- 3.1.3. El Administrador selecciona la opción “Generar”. Ver flujo alternativo 3.3.1.
- 3.1.4. El sistema ejecuta el procedimiento del algoritmo genético.
- 3.1.5. El sistema muestra los horarios académicos para cada uno de los semestres y habilita la opción “Guardar”.

- 3.1.6. El Administrador visualiza los horarios académicos generados y selecciona la opción “Guardar”. Ver flujo alternativo 3.3.1.
- 3.1.7. El sistema graba los horarios académicos y muestra un mensaje de información.
- 3.1.8. El Administrador acepta el mensaje mostrado.
- 3.1.9. El Administrador selecciona la opción “Salir”.
- 3.1.10. El sistema cierra la pantalla “Generar Horario Académico”.
- 3.1.11. El caso de uso termina.

3.2. Sub Flujos

No aplica.

3.3. Flujos alternativos

- 3.3.1 En los pasos 3.1.3 y 3.1.6 del flujo básico.

Si el Administrador selecciona la opción “Cancelar”.

- 1. Continúa en el paso 3.1.2 del flujo básico.

Si el Administrador selecciona la opción “Salir”.

- 1. Continúa en el paso 3.1.10 del flujo básico.

4. POSCONDICIONES

La Información de los horarios académicos semestrales está almacenada en el sistema.

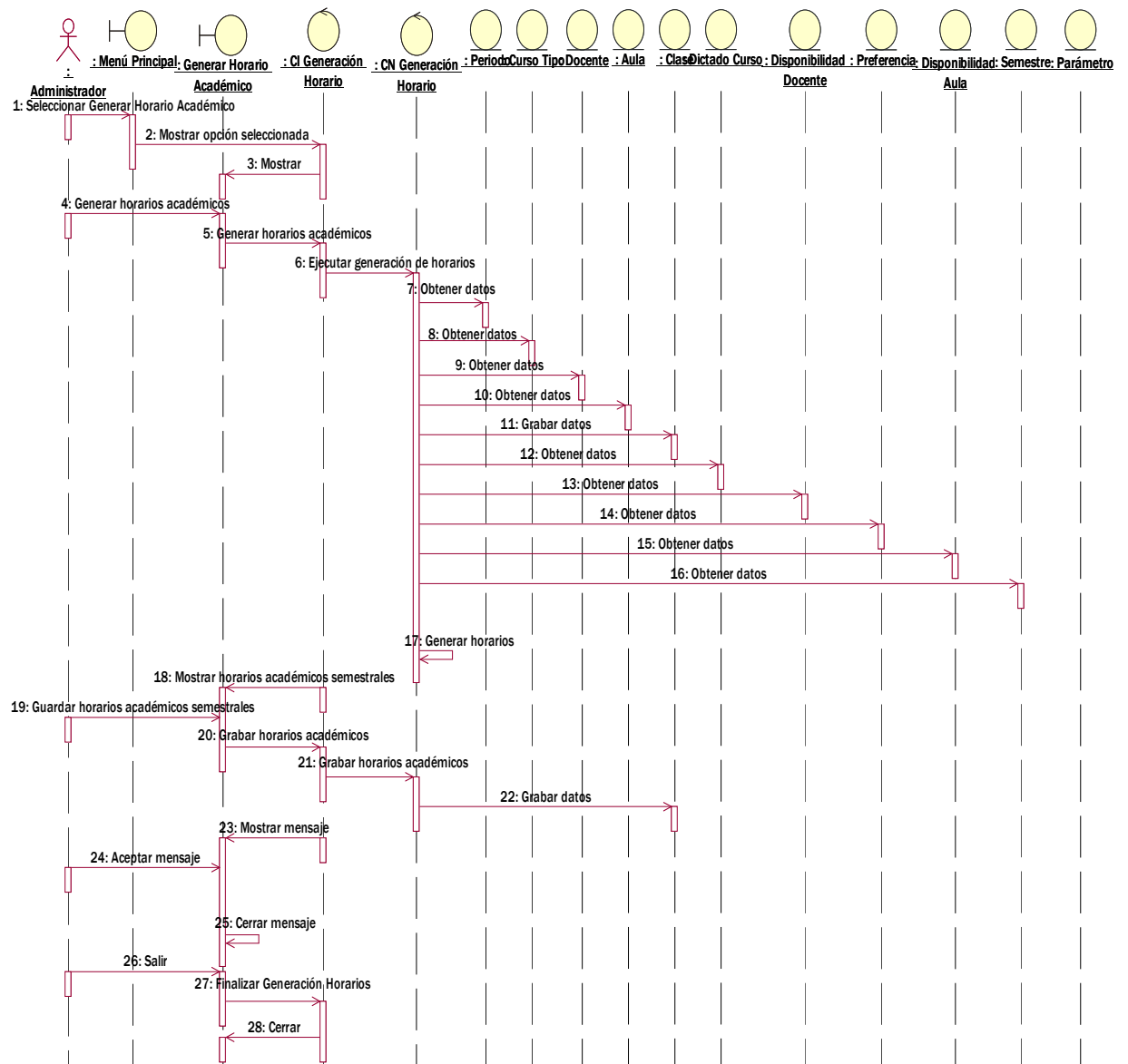


Figura 5.15. Diagrama de Secuencia de “Generar Horario Académico”

5.3. Diagrama de Clases

Con este diagrama describiremos la estructura del sistema mostrando para ello las clases, los atributos y las relaciones que existen entre ellos. Cada clase posee propiedades ó atributos y un conjunto de operaciones. A continuación presentaremos las clases, sus propiedades y las relaciones entre dichas clases.

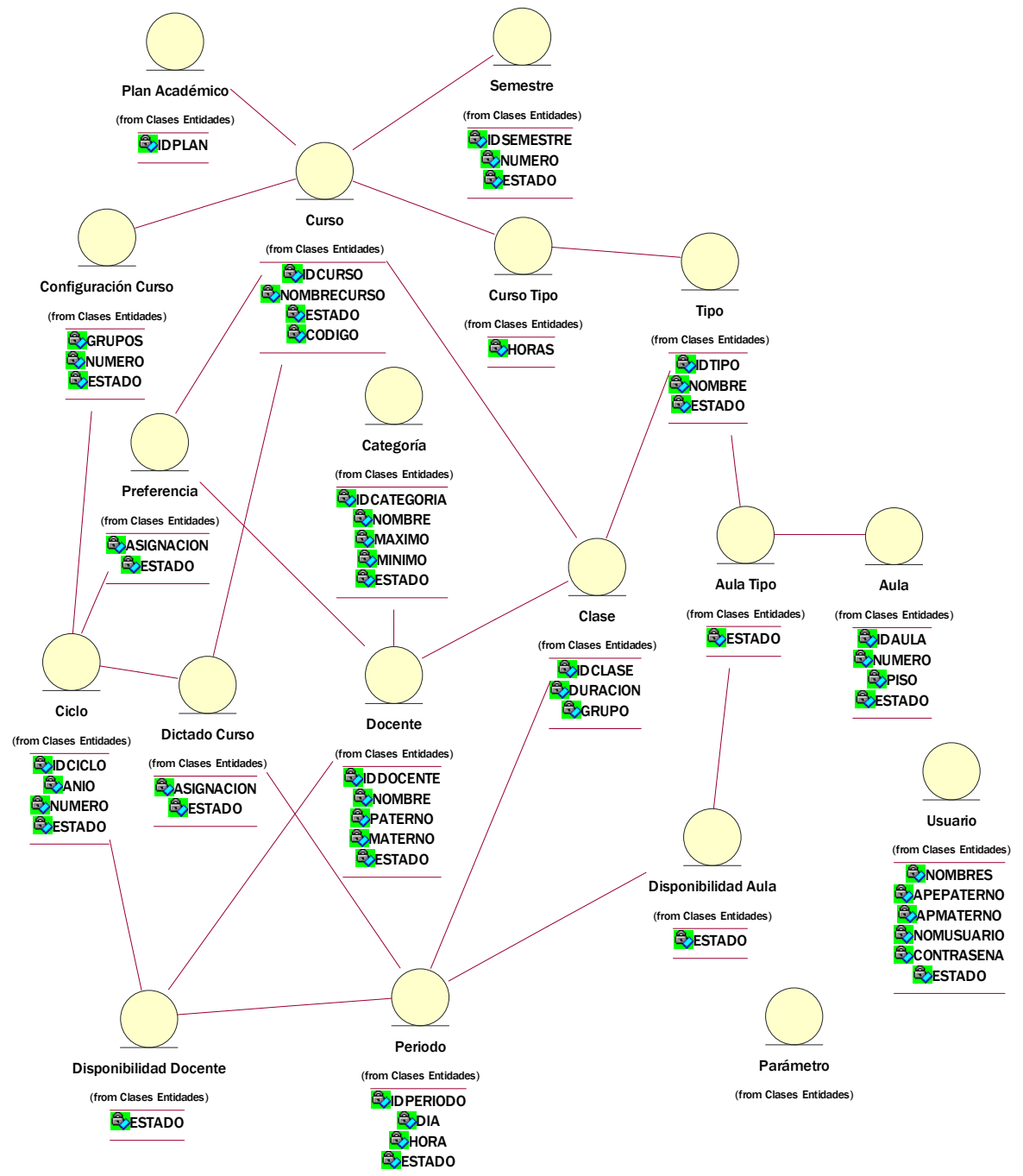


Figura 5.16. Diagrama de clases y sus relaciones

Entre las entidades mostradas tenemos las siguientes:

- **Ciclo:** Un ciclo académico corresponde a un rango de meses en un año para la cual se destina la enseñanza a los alumnos. Cada año consta de tres ciclos académicos, que inicia en el ciclo cero y termina en el ciclo dos.

- **Semestre:** Un semestre académico corresponde a la división en periodos que se realiza para cumplir con la enseñanza de todos los cursos académicos que establece la carrera profesional. Son diez el número de semestres académicos. Cada semestre está conformado por un conjunto de cursos, los cuales se encuentran debidamente agrupados.
- **Plan de Estudios:** Un plan de estudios está conformado por todos los cursos que determina la carrera profesional.
- **Periodo:** El periodo corresponde a cada hora académica que existe en una semana.
- **Curso:** Es la unidad educativa correspondiente a una materia que contempla una duración de instrucción.
- **Tipo:** Corresponde a una forma de clasificación y pueden ser teoría, práctica y laboratorio.
- **Curso Tipo:** Se refiere a la correspondencia existente entre curso y tipo anteriormente definidos. Un curso puede estar asociado a uno ó más tipos; tales como curso 1 de tipo A y tipo B, curso 2 de tipo A, curso 3 de tipo B, etc.
- **Configuración Curso:** Se establece la cantidad de grupos que se van a aperturar por cada curso y el número de personas que pueden matricularse por grupo aperturado.
- **Dictado Curso:** Indica los periodos en que puede realizarse el dictado de un determinado curso. Puede ó no existir una asignación fija de un periodo para un curso específico.
- **Docente:** Persona que realiza el proceso de enseñanza de cursos en la facultad.
- **Categoría:** Es la forma en que se clasifican los docentes, según la cantidad de horas de enseñanza y la modalidad de tipo de docente.

- **Preferencia:** Representa la relación existente entre cursos y docentes, ya que un docente tiende a dictar uno ó más cursos según su ámbito de enseñanza.
- **Disponibilidad Docente:** Representa la relación existente entre periodos y docentes. Un docente tiene una disponibilidad laboral en la cual puede realizar el dictado de clases.
- **Aula:** Cada uno de los ambientes de la facultad en los cuales se realiza el dictado de los cursos tanto de los tres tipos establecidos.
- **Aula Tipo:** Se refiere a la correspondencia existente entre aula y tipo. Cada aula está asociada a un tipo, el cual determina las condiciones que pueda tener el aula para realizar el dictado de las clases. Así tenemos el aula 101 de tipo teoría, el aula 301 de tipo laboratorio, etc.
- **Disponibilidad Aula:** Representa la relación existente entre periodos y aulas. Un aula tiene una disponibilidad a la semana en la cual se puede realizar el dictado de clases.
- **Usuario:** Se refiere a los datos de la persona que puede acceder al sistema considerándole ciertos privilegios de accesibilidad.
- **Parámetro:** Se establece una configuración inicial para la ejecución del proceso de generación de horarios académicos semestrales; tales como tamaño de la población, número de generaciones, etc.

5.4. Modelo de Datos

El modelo de datos que vamos a establecer corresponde a un esquema conceptual que nos permitirá representar los datos que se pretenden almacenar en la base datos. Se muestran las entidades con sus respectivos atributos, además de poseer una clave que le permite identificarse de manera única; a su vez se muestran las conexiones entre dichas entidades.

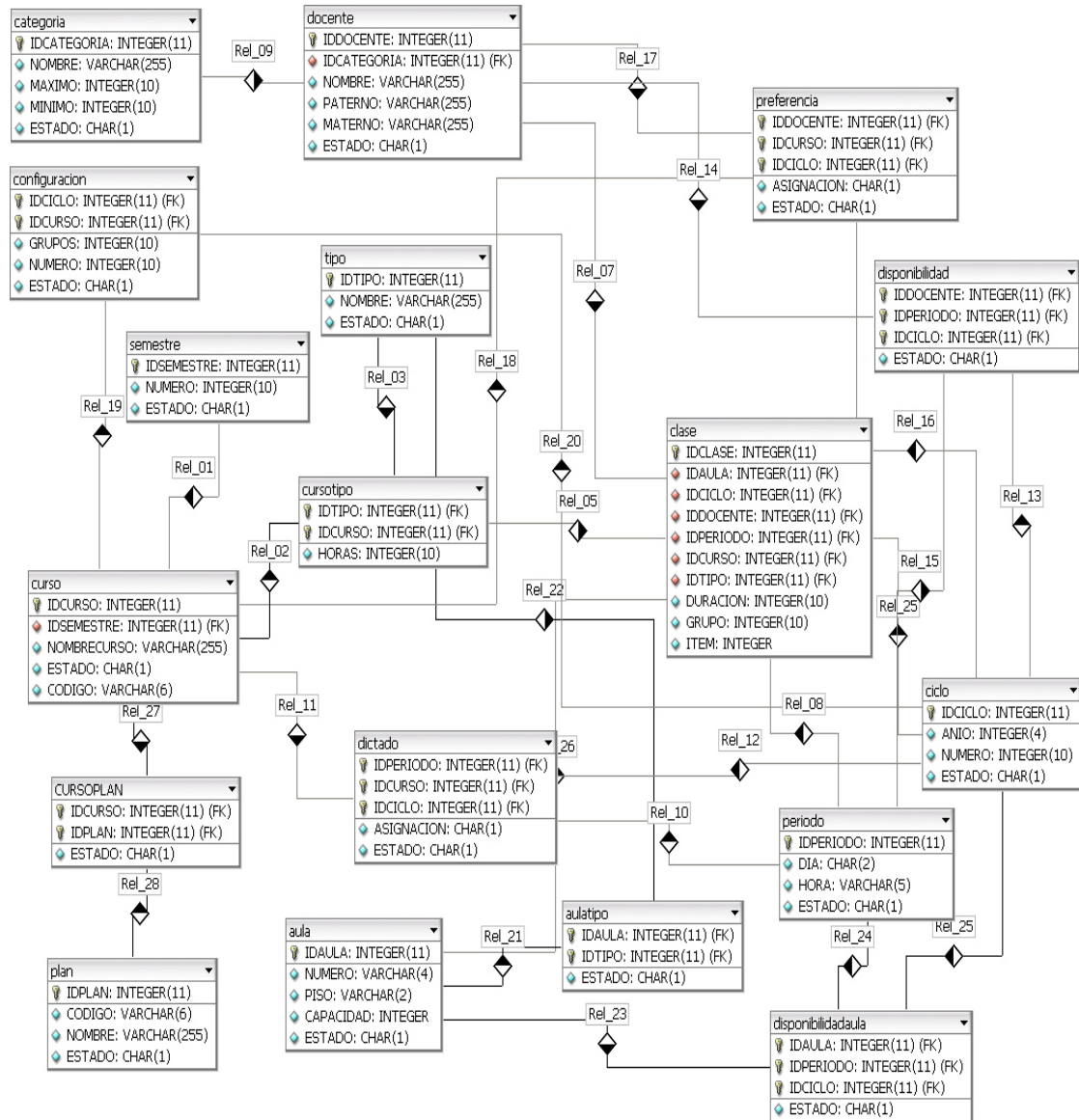


Figura 5.17. Diagrama Entidad – Relación

5.5. Arquitectura de la Solución

La arquitectura del software que se plantea nos proporcionará un marco de referencia para una posterior implementación de la solución. El diseño de la arquitectura se realizará en base a los objetivos y restricciones del sistema.

La arquitectura mostrada a continuación representa a los componentes involucrados en la solución, y se divide en niveles para facilitar su respetiva comprensión y separar sus funcionalidades.

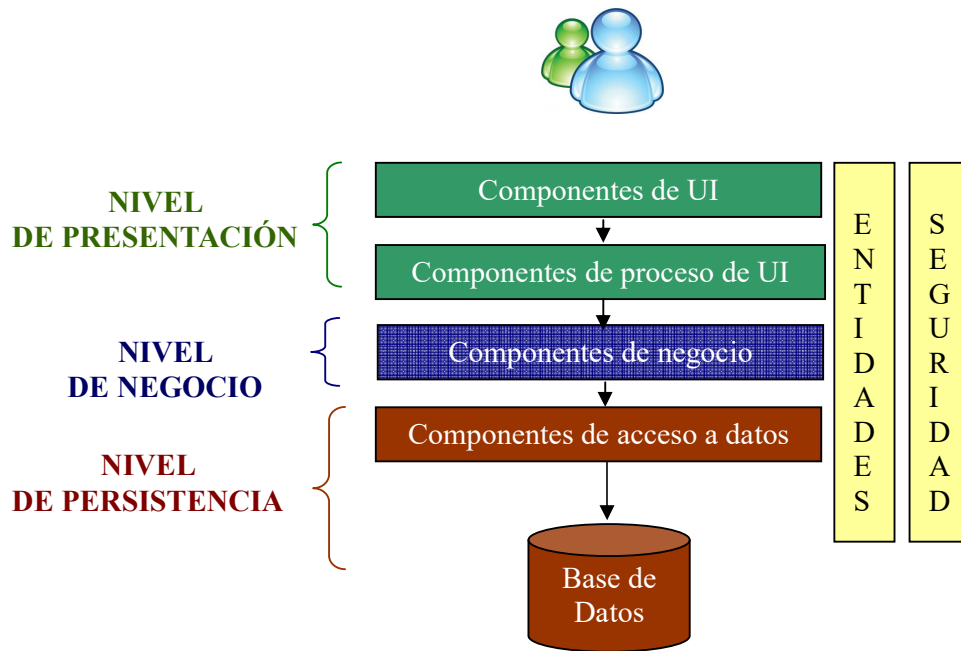


Figura 5.18. Arquitectura del Sistema

Esta arquitectura presenta los siguientes niveles: el nivel de presentación, de negocio y de persistencia.

- El nivel de presentación está compuesto por los componentes de UI, que son las interfaces gráficas que permiten al usuario interactuar con el sistema; y los componentes de proceso de UI que permiten una adecuada administración de las interfaces.
- El nivel de negocio está compuesto por los componentes de negocio, que es la capa que presenta toda la lógica del negocio que se está implementando.
- El nivel de persistencia está compuesto por los componentes de acceso a datos que permiten interactuar con las entidades para obtener la información de la base de datos.

Además paralelo a todos estos niveles tenemos las entidades que resultan ser el enlace entre cada una de las capas, y además la seguridad de los datos en todos los niveles.

5.5.1. Restricciones no funcionales

Se han identificado algunos requerimientos no funcionales que definen las metas y restricciones arquitectónicas:

- El lenguaje de programación a emplearse para el desarrollo de la aplicación será el lenguaje Java. Es un lenguaje orientado a objetos, los cuales son elementos básicos para modelar los datos sobre los que trabajará nuestra solución.
- El servidor de Base de Datos que se va a utilizar es MySQL 5.1.41. Es un gestor para base de datos relacionales cuyo objetivo es velocidad al realizar las operaciones y robustez al asegurar una integridad completa de los datos.
- La herramienta de desarrollo es NetBeans 6.5

5.6. Vista de Componentes

A través de la vista de componentes podemos describir los componentes que constituyen el sistema y sus dependencias. Cada componente representará un elemento físico que está incluido en la construcción de la aplicación; mientras que las relaciones establecidas nos indicarán que un componente utiliza los servicios ofrecidos por otro componente.

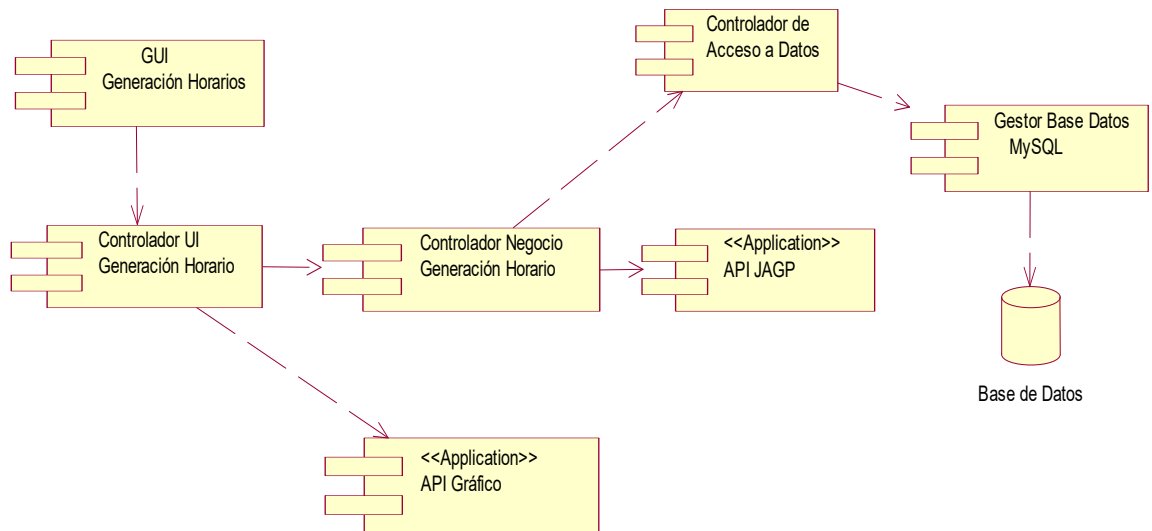


Figura 5.19. Diagrama de Componentes

En este diagrama, podemos observar que cada uno de los componentes son partes modulares, desplegables y reutilizables del sistema. Entre ellos tenemos: El componente GUI Generación de Horarios (Interfaz Gráfica de Usuario de Generación de Horarios) que está formada por todos las interfaces gráficas, el componente Controlador UI Generación Horarios (Controlador de Interfaz de Usuario de Generación de Horarios), el componente Controlador de Negocio de

Generación de Horarios, el componente Controlador de Acceso a Datos, el componente Gestor de Base de Datos, el componente JGAP y el componente Gráfico.

En esta parte vamos a detallar el componente JGAP para conocer sus respectivas funcionalidades que ayudan al sistema en la generación de horarios académicos.

JGAP son las iniciales de “Java Genetic Algorithms Package”, es un paquete de Programación Genética y Algoritmos Genéticos escrito en Java. Proporciona los mecanismos básicos de genética que puede ser fácilmente utilizado para aplicar los principios evolutivos a la solución de problemas. JGAP fue diseñado para ser muy fácil de usar, es decir, está listo para usarse; mientras que también está diseñado para ser altamente modular, de forma que se pueda fácilmente personalizar los operadores genéticos y otros sub-componentes.

A continuación mostraremos la lista de paquetes de JGAP, y para mayor detalle ver el apéndice 4:

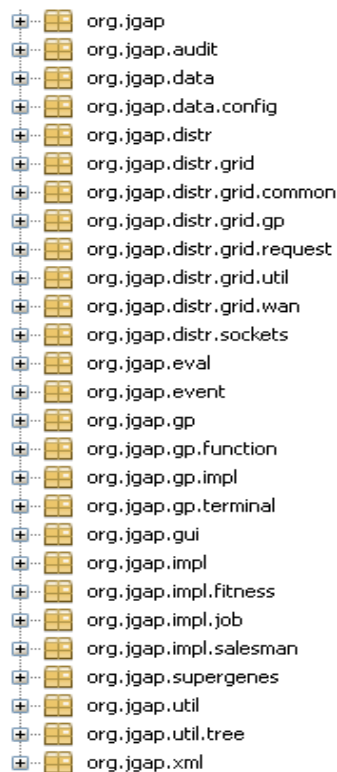


Figura 5.20. Estructura de paquetes de JGAP 3.4.4

5.7. Vista de Despliegue

La vista de despliegue nos permite determinar en donde se ejecutan los componentes del software anteriormente descritos. De esta manera podemos

establecer la disposición física de los nodos que son parte del sistema, es decir, vamos a modelar la topología del hardware donde se va a ejecutar el sistema.

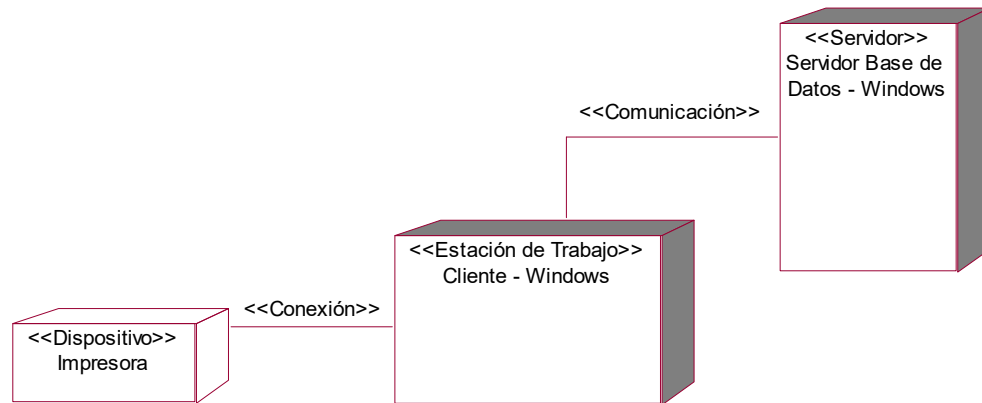


Figura 5.21. Diagrama de Despliegue

El diagrama de despliegue mostrado representa la arquitectura física de la solución, en el cual se pueden observar 3 nodos: Impresora, Estación Cliente y Servidor de Base de Datos.

5.8. Prototipos de la Solución

Para establecer el diseño de las pantallas prototipo de la solución se utilizará la información reunida en el análisis, con la ayuda de éste se podrá determinar cómo construir una interfaz gráfica de usuario que satisfaga sus necesidades y a su vez lograr que la aplicación sea flexible, coherente, eficiente y sencilla de utilizar.

El objetivo de esta sección es especificar cada formato individual de la interfaz de las pantallas del sistema. En la definición de cada pantalla se considerarán aquellos aspectos que son importantes para el diseño y construcción del sistema.

5.8.1. Ventana de Ingreso al Sistema

Esta pantalla hace referencia al Caso de Uso “Ingresar al Sistema”, a través del cual el usuario ingresará los datos necesarios para poder ingresar al sistema.

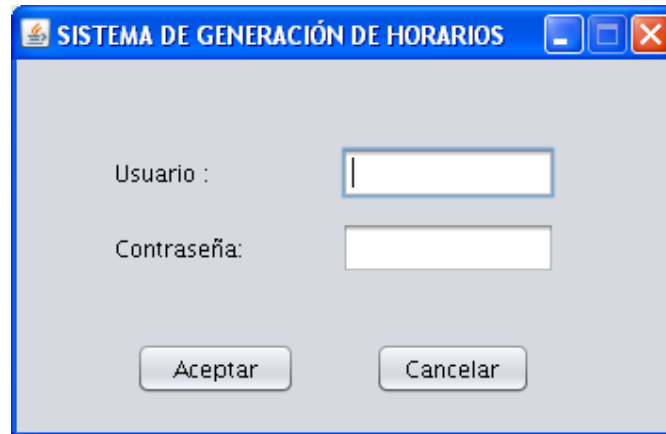
Una ventana de software con un título azul que dice "SISTEMA DE GENERACIÓN DE HORARIOS". En el interior, hay un campo de texto etiquetado "Usuario :" y otro etiquetado "Contraseña:". Debajo de estos campos hay dos botones: "Aceptar" y "Cancelar".

Figura 5.22. Prototipo del Caso de Uso “Ingresar al Sistema”

Luego de haber ingresado los datos del usuario, se mostrará la pantalla principal del sistema con un menú de opciones, por medio del cual el usuario podrá interactuar con el sistema.

La siguiente pantalla hace referencia al Menú Principal del Sistema, a través del cual el usuario podrá visualizar las tareas que puede realizar de acuerdo a su perfil establecido.

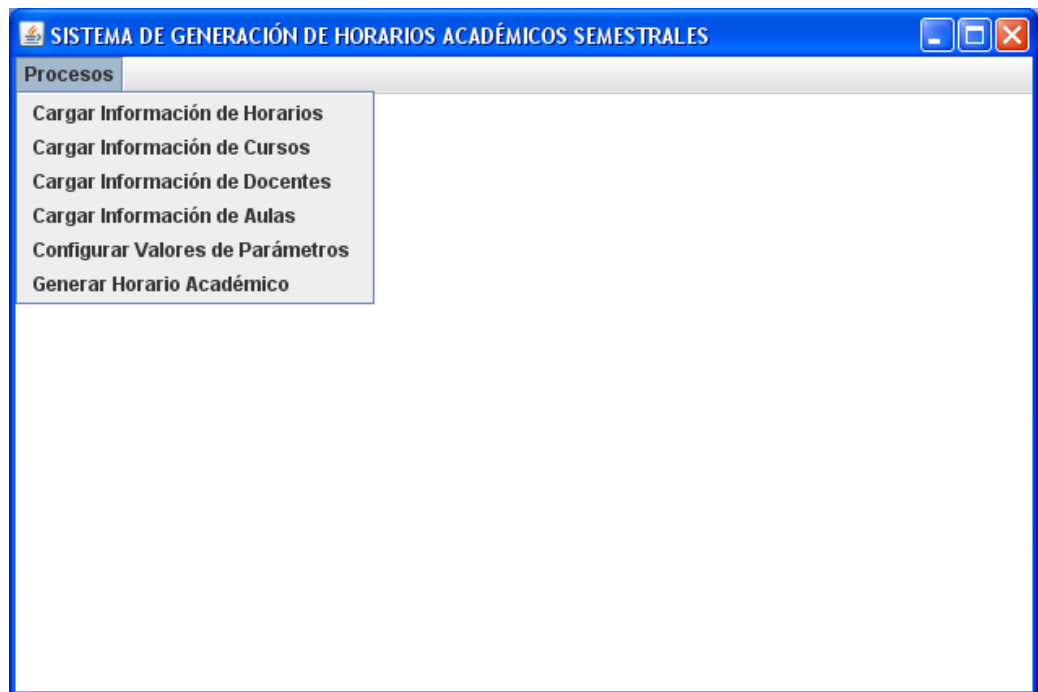
Una ventana de software con un título azul que dice "SISTEMA DE GENERACIÓN DE HORARIOS ACADÉMICOS SEMESTRALES". Debajo del título hay una pestaña etiquetada "Procesos". A la izquierda de la ventana principal hay un menú desplegable con las siguientes opciones: "Cargar Información de Horarios", "Cargar Información de Cursos", "Cargar Información de Docentes", "Cargar Información de Aulas", "Configurar Valores de Parámetros" y "Generar Horario Académico".

Figura 5.23. Prototipo del Menú Principal

5.8.2. Ventana de Carga de Información de Horarios

Esta pantalla hace referencia al Caso de Uso “Cargar Información de Horarios”, a través del cual el usuario ingresará los periodos en que se puede realizar el dictado de clases a la semana.

La imagen muestra un prototipo de una ventana de software con el título "CARGAR INFORMACIÓN DE HORARIOS". Dentro de la ventana, hay un encabezado que dice "Seleccionar horas de dictado de clases en la semana". Debajo de este encabezado hay una tabla con 7 columnas: "Hora", "Lunes", "Martes", "Miércoles", "Jueves", "Viernes" y "Sábado". La tabla tiene 12 filas vacías para ingresar datos. En la parte inferior de la ventana, hay dos botones: "Guardar" y "Cancelar".

Hora	Lunes	Martes	Miércoles	Jueves	Viernes	Sábado

Figura 5.24. Prototipo del Caso de Uso “Cargar Información de Horarios”

5.8.3. Ventana de Carga de Información de Cursos

La pantalla hace referencia al Caso de Uso “Cargar Información de Cursos”, a través del cual el usuario ingresará los cursos que pertenecen a un plan de estudios y el horario en que pueden ser dictados dichos cursos. Además se detallará la cantidad de horas que se dicta por cada uno de los tipos relacionados al curso.

La imagen muestra un prototipo de una ventana de software titulada "CARGAR INFORMACIÓN DE CURSOS". La interfaz está organizada en varias secciones:

- Sección superior:** Incluye un menú desplegable "Plan Académico" con la opción "2010 - I" seleccionada, un campo de texto "Ruta Archivo:" y un botón "Buscar". Debajo de estos hay dos botones: "Cargar" y "Limpiar".
- Sección central izquierda:** Un panel de árbol de archivos (JTree) que muestra una estructura de carpetas: "JTree", "colors", "sports" y "food".
- Sección central derecha:** Contiene un menú desplegable "Tipo:" con "Teoría" seleccionado, un campo de texto "Horas de dictado:" con el valor "0" y un botón de flechas para navegar. Debajo de esto, un subencabezado "Horario Dictado:" precede a una tabla de horarios.
- Tabla de Horario Dictado:** Es una tabla con 7 columnas encabezadas por los días de la semana: "Hora", "Lunes", "Martes", "Miérco...", "Jueves", "Viernes" y "Sábado". El cuerpo de la tabla está vacío, listo para ser llenado.
- Sección inferior:** Incluye un checkbox "Asignación Previa" y una fila de botones de acción: "Nuevo", "Modificar", "Eliminar", "Guardar" y "Cancelar".

Figura 5.25. Prototipo del Caso de Uso “Cargar Información de Cursos”

5.8.4. Ventana de Carga de Información de Docentes

La pantalla hace referencia al Caso de Uso “Cargar Información de Docentes”, a través del cual el usuario ingresará la lista de docentes, incluyendo los cursos que dictan y su disponibilidad horaria.

La imagen muestra un prototipo de una ventana de software titulada "CARGAR INFORMACIÓN DE DOCENTES". La interfaz está organizada en varias secciones:

- Cargar Archivo:** Incluye un campo de texto "Ruta Archivo:" con un botón "Buscar" a su derecha. Debajo del campo hay dos botones: "Guardar" y "Limpiar".
- Árbol de Archivos (JTree):** Se encuentra a la izquierda, mostrando una estructura de carpetas con los nombres "colors", "sports" y "food".
- Cursos de Dictado:** Esta sección está dividida en dos paneles:
 - Listado de Cursos:** Contiene una lista de "Item 1" hasta "Item 5".
 - Cursos Seleccionados:** También contiene una lista de "Item 1" hasta "Item 5".
 - Entre los paneles hay dos botones: "Agregar >>" y "<< Quitar".
- Disponibilidad Horaria:** Se trata de una tabla con 7 columnas encabezadas por los días de la semana: "Lunes", "Martes", "Miércoles", "Jueves", "Viernes" y "Sábado". La primera columna está etiquetada como "Hora". La tabla tiene 10 filas vacías para ingresar datos.
- Botones de Acción:** En la parte inferior de la ventana, hay una fila de botones: "Nuevo", "Modificar", "Eliminar", "Aceptar" y "Cancelar".

Figura 5.26. Prototipo del Caso de Uso “Cargar Información de Docentes”

5.8.5. Ventana de Carga de Información de Aulas

La pantalla hace referencia al Caso de Uso “Cargar Información de Aulas”, a través del cual el usuario ingresará la lista de aulas, incluyendo el tipo de aula, la capacidad y su disponibilidad durante la semana.

La imagen muestra un prototipo de una ventana de software titulada "CARGAR INFORMACIÓN DE AULAS". La ventana tiene una barra de título azul con botones de control estándar. El contenido principal está dividido en varias secciones:

- Cargar Archivo:** Incluye un campo de texto "Ruta Archivo:" con un botón "Buscar" a su derecha. Debajo del campo hay dos botones: "Cargar" y "Limpiar".
- Datos Generales:** Contiene un campo "Tipo:" con un menú desplegable que muestra "Teoría", y un campo "Capacidad:" con el valor "50".
- Horarios Disponibles:** Presenta una tabla con 7 columnas encabezadas por los días de la semana: "Hora", "Lunes", "Martes", "Miércoles", "Jueves", "Viernes" y "Sábado". La tabla tiene 10 filas vacías para ingresar los horarios.
- Área de Navegación:** A la izquierda de la tabla hay un panel con una estructura de árbol que muestra carpetas como "JTree", "colors", "sports" y "food".
- Botones de Acción:** En la parte inferior de la ventana hay dos grupos de botones. El primer grupo contiene "Nuevo", "Modificar" y "Eliminar". El segundo grupo contiene "Guardar" y "Cancelar".

Figura 5.27. Prototipo del Caso de Uso “Cargar Información de Aulas”

5.8.6. Ventana de Configuración de Valores de Parámetros

La pantalla hace referencia al Caso de Uso “Configurar Valores de Parámetros”, a través del cual el usuario ingresará el tamaño de la población y el número de generaciones que son los parámetros para dar inicio al proceso de generación de horarios de clases.

Esta es una ventana de configuración con el título "CONFIGURAR VALORES DE PARÁMETROS". Contiene dos campos de entrada: "Tamaño de la población:" y "Número de generaciones:". Debajo de estos campos hay dos botones: "Guardar" y "Cancelar". En la parte inferior de la ventana hay una tabla con tres columnas: "Nº", "Tamaño de Población" y "Nº de Generaciones". La tabla tiene tres filas vacías para ingresar datos.

Nº	Tamaño de Población	Nº de Generaciones

Figura 5.28. Prototipo del Caso de Uso “Configurar Valores de Parámetros”

5.8.7. Ventana de Generación de Horario Académico

La pantalla hace referencia al Caso de Uso “Generar Horario Académico”, a través del cual el usuario generará el horario académico de cada uno de los semestres, con la configuración de parámetros establecida y los datos ingresados sobre los docentes, las horas académicas y las aulas.

Esta es una ventana con el título "GENERAR HORARIO ACADÉMICO". Contiene una sección titulada "Horario Académico por Semestre" que muestra una grilla de botones para seleccionar un semestre. Los botones están organizados en dos filas: la primera fila contiene "Semestre VI", "Semestre VII", "Semestre VIII", "Semestre IX" y "Semestre X"; la segunda fila contiene "Semestre I", "Semestre II", "Semestre III", "Semestre IV" y "Semestre V". Debajo de la grilla hay un área grande y vacía para mostrar el horario generado. En la parte inferior de la ventana hay tres botones: "Generar", "Cancelar" y "Guardar".

Figura 5.29. Prototipo del Caso de Uso “Generar Horario Académico”

Capítulo 6 : Pruebas y Resultados

En este capítulo se mostrarán las pruebas que se realizaron para determinar la selección de un método entre diversos métodos de operadores genéticos; así como también escoger una serie de parámetros de configuración que permitirán al algoritmo genético poder dar inicio a la búsqueda de una solución óptima. Además se presentarán los resultados que fundamentan nuestro análisis para la selección de las mejores opciones, de tal manera que pueda darse solución al problema de generación de horarios de clases.

6.1. Introducción

Como objetivo del presente trabajo se propuso encontrar una solución aproximada, a través de algoritmos genéticos, al problema de generación de horarios para la Escuela Académica Profesional de Ingeniería de Sistemas e Informática de la Facultad de Ingeniería de Sistemas e Informática de la Universidad Nacional Mayor de San Marcos.

Ahora que ya hemos implementado el algoritmo, el siguiente paso será proceder con su respectiva evaluación. Esta evaluación se realizará en dos partes:

1. La primera, constará en encontrar los mejores métodos para cada etapa del algoritmo. Como ya vimos en el capítulo dos, el algoritmo genético presenta etapas tales como selección, emparejamiento, mutación y reinserción; por lo tanto debemos identificar el mejor método que nos permite realizar adecuadamente el correspondiente proceso en cada una de dichas etapas.
2. La segunda, se enfocará a encontrar la mejor configuración para cada método que se haya seleccionado en la primera parte, de tal manera que dicha configuración nos aporte una mejor performance y mejores resultados para nuestro problema.

6.2. Casos de Análisis

Contaremos con dos casos de análisis diferentes para esta sección, ambos se basarán en la información proporcionada que corresponde al ciclo 2009-I de la Escuela Académica Profesional de Ingeniería de Sistemas e Informática, la cual fue elaborada manualmente por el personal responsable.

Los casos de análisis serán descritos a continuación:

1. El primer caso de análisis, tomará en cuenta el 30% de la información necesaria para realizar la generación de horarios del ciclo académico en cuestión. Este porcentaje de la información corresponde a cursos que son dictados hasta el tercer semestre.

A continuación detallamos los recursos tomados en cuenta para este caso:

- Se cuenta con un total de 18 cursos, con los respectivos rangos de periodos en los cuales podrían ser dictados.
 - Se cuenta con 45 docentes de diferentes categorías, con sus respectivas disponibilidades y preferencia de dictado.
 - Se cuenta con 11 aulas con sus respectivas disponibilidades de horario.
2. El segundo caso de análisis, tomará en cuenta el 70% de la información necesaria para realizar la generación de horarios del ciclo académico en cuestión. Este porcentaje de la información corresponde a cursos que son dictados hasta el séptimo semestre.

A continuación detallamos los recursos tomados en cuenta para este caso:

- Se cuenta con un total de 42 cursos, con los respectivos rangos de periodos en los cuales podrían ser dictados.
- Se cuenta con 76 docentes de diferentes categorías, con sus respectivas disponibilidades y preferencia de dictado.
- Se cuenta con 20 aulas con sus respectivas disponibilidades de horario.

3. El tercer caso de análisis, tomará en cuenta el 100% de la información necesaria para realizar la generación de horarios del ciclo académico en cuestión.

A continuación detallamos los recursos tomados en cuenta para este caso:

- Se cuenta con un total de 67 cursos, con los respectivos rangos de periodos en los cuales podrían ser dictados.
- Se cuenta con 101 docentes de diferentes categorías, con sus respectivas disponibilidades y preferencia de dictado.
- Se cuenta con 25 aulas con sus respectivas disponibilidades de horario.

Los dos primeros casos de análisis serán empleados en la sección siguiente que corresponde al Esquema de Pruebas. Cada caso será utilizado en puntos distintos de dicha sección.

El último caso será para mostrar un horario completo desarrollado íntegramente por nuestro algoritmo.

6.3. Esquema de Pruebas

Para medir el desempeño del algoritmo genético en base a sus diferentes variaciones, como por ejemplo usar un tipo de operador u otro, utilizaremos tres tipos de mediciones diferentes:

- El número de generaciones. Se medirá la cantidad de generaciones promedio que se necesita para lograr una solución óptima.
- La frecuencia de éxito. Se identificará la frecuencia con la cual se obtienen resultados satisfactorios en las diferentes ejecuciones del algoritmo.

La cantidad de veces que se ejecutará el algoritmo para calcular el promedio de número de generaciones, la frecuencia de éxito resultante y el número promedio de individuos diferentes será de 40.

Los dos casos de análisis descritos en la sección anterior denominada Casos de Análisis, serán empleados en los puntos siguientes de esta sección. El primer caso de análisis se usará en los puntos 6.3.1, 6.3.2 y 6.3.3; mientras que para el punto 6.3.4 se usará el segundo caso de análisis.

Otro punto importante es saber que para los puntos 6.3.1, 6.3.2 y 6.3.3 se utilizará como fase representativa a la segunda ya que, como se mencionó anteriormente, es la que presenta un mayor grado de complejidad. Los resultados obtenidos de estas pruebas serán empleados para las demás fases. Y para el punto 6.3.4 se realizarán pruebas individuales para cada una de las fases.

A continuación se dará inicio a la primera parte de la evaluación que se encargará de encontrar los mejores métodos de cada etapa del algoritmo. Empezaremos con el proceso de selección, luego seguiremos con cada uno de los operadores genéticos, y por último, con el proceso de reinserción.

6.3.1. Pruebas sobre el proceso de selección

En esta parte veremos las pruebas realizadas para escoger el método de selección más apropiado para nuestro problema; además de ver si es conveniente ó no escalar las aptitudes resultantes de los cromosomas antes de que sean evaluadas en el proceso de selección.

6.3.1.1. Prueba de escalamiento de la aptitud

Según lo visto en el capítulo 4, se podría dar el caso que la aparición de un súper-individuo nos haga llegar a óptimos locales, impidiendo al algoritmo poder avanzar. Para evitar esto se planteó usar el método de escalamiento de la aptitud para disminuir la diferencia relativa de aptitudes entre individuos y así disminuir la probabilidad de que aparezcan súper-individuos.

En este escenario pasaremos a probar el escalamiento lineal que se expresa mediante la formula: $f' = a + b*f$ usando los valores a continuación:

a	b
0	1
0.1	0.9
0.2	0.8
0.3	0.7
0.4	0.6
0.5	0.5
0.6	0.4
0.7	0.3
0.8	0.2
0.9	0.1

Tabla 6.1. Valores para realizar el escalamiento de la aptitud

En la Figura 6.1, se muestra el número de generaciones que necesita el algoritmo para obtener una solución óptima, de acuerdo al uso de diferentes valores de escalamiento. Se puede observar que los valores de escalamiento $a = 0.2$, $b = 0.8$ y $a = 0$, $b = 1$ (lo cual implica que realmente no se escala la aptitud original) dan resultados iguales, lo cual nos obliga a usar el segundo criterio de evaluación que definimos.

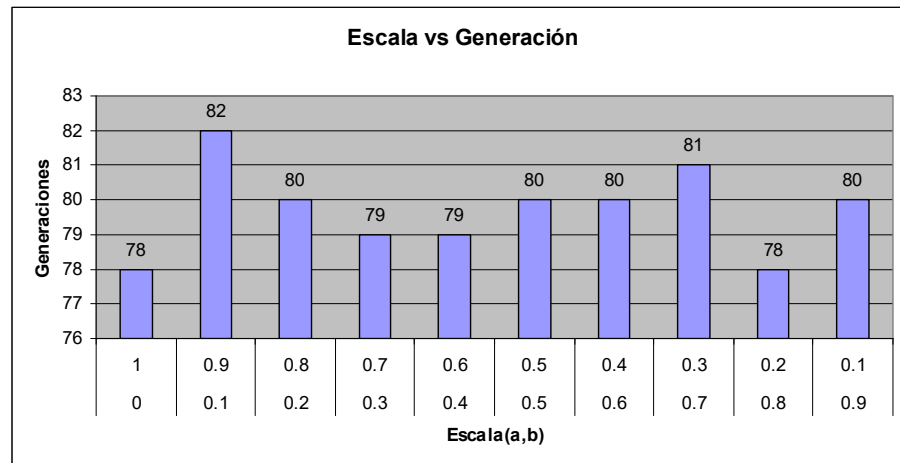


Figura 6.1. Número de generaciones vs. Escalamiento de la aptitud utilizado

La Figura 6.2 muestra la frecuencia de éxito en encontrar una solución óptima al problema, mostrándonos claramente que la escala $a = 0$ y $b = 1$ posee un valor del 95% superando al resto de escalas.

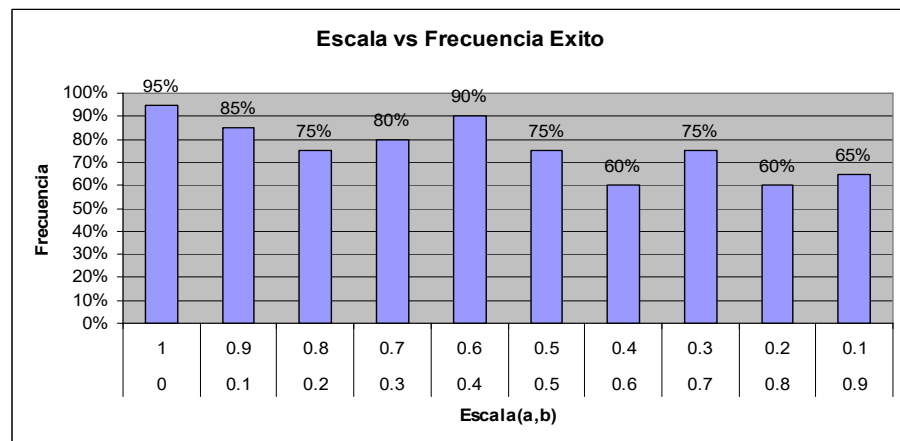


Figura 6.2. Frecuencia de éxito vs. Escalamiento de la aptitud utilizado

La interpretación de tener como mejor resultado a la escala $a = 0$ y $b = 1$ es que no se debería de escalar la aptitud original del individuo, y esto se da debido a que la forma de penalizar la aparición de errores por la que se ha optado no permite que la diferencia relativa entre todos los individuos de la población sea grande, como para admitir la aparición de súper-individuos que acaparen dicha población.

6.3.1.2. Prueba sobre el método de selección

Para comprobar cuál método de selección se comporta mejor para nuestro problema, se compararon dos de los más populares, el de ruleta y el de torneo, los cuales fueron mencionados en el capítulo 2. Obteniéndose los siguientes resultados que son mostrados en la Figura 6.3.

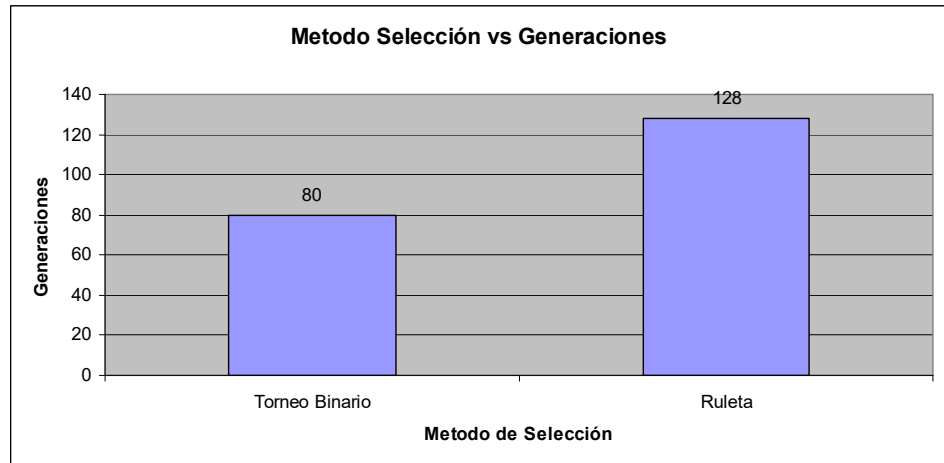


Figura 6.3. Número de generaciones vs. Método de selección utilizado

Como se puede observar el uso del método de torneo supera ampliamente al método de ruleta, ya que este último necesita un número mayor de generaciones para encontrar una solución factible al problema.

6.3.1.3. Prueba sobre el tamaño del torneo

Otro punto importante es calcular el tamaño del torneo que vamos a usar, dado que éste afecta directamente al desempeño del algoritmo. En la Figura 6.4 vemos que se ha realizado pruebas con diferentes tamaños de torneo, y a simple vista pareciera que a mayor tamaño de torneo el algoritmo encuentra una solución de manera mas rápida. Pero si contrastamos estos resultados con la Figura 6.5 en la cual se muestra la frecuencia de éxito para cada tamaño de torneo, se puede llegar a otra conclusión.

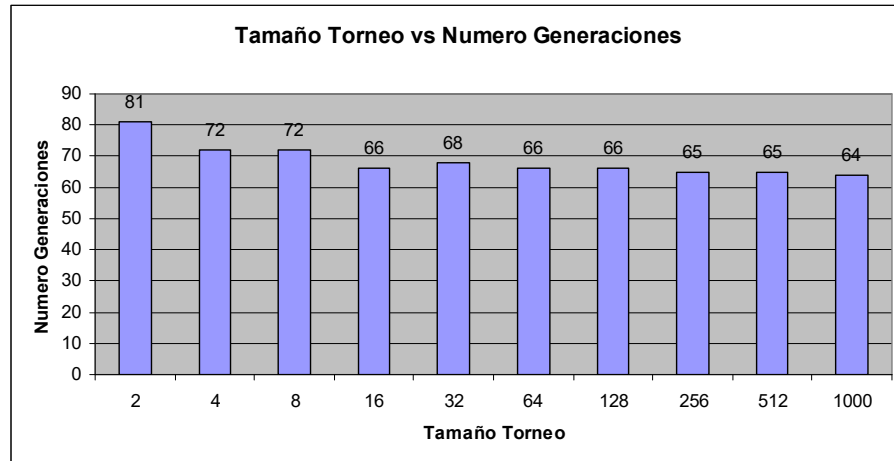


Figura 6.4. Número de generaciones vs. Tamaño de torneo utilizado

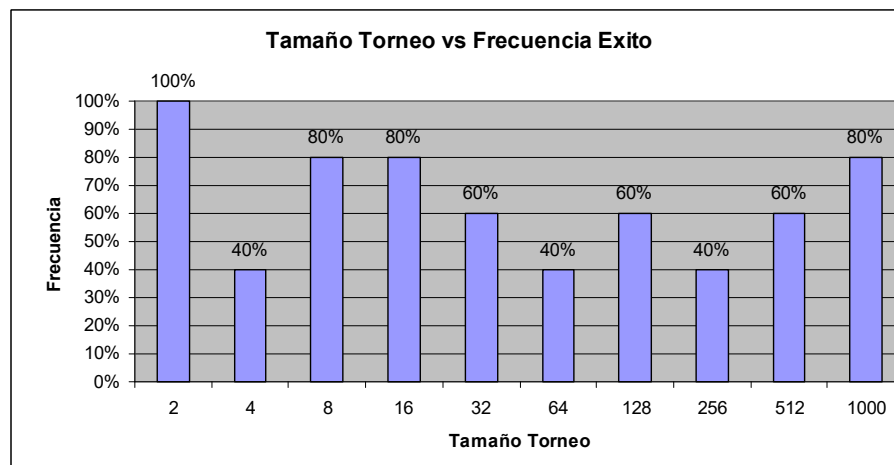


Figura 6.5. Frecuencia de éxito vs. Tamaño de torneo utilizado

Usando la información de ambas gráficas escogemos el tamaño de torneo más adecuado que es el de valor dos; dado que los otros, a pesar de llegar a una solución en un número menor de generaciones, presentan una frecuencia mucho menor para alcanzar soluciones factibles.

6.3.2. Pruebas sobre los operadores genéticos

En esta parte veremos las pruebas realizadas para escoger el mejor operador de emparejamiento y el mejor operador de mutación que sean adecuados para nuestro problema, así como el porcentaje necesario para que los respectivos operadores seleccionados tengan un mejor desempeño.

6.3.2.1. Prueba sobre el método de emparejamiento

Para escoger el operador de emparejamiento más adecuado, se realizaron pruebas con cuatro métodos: emparejamiento de un punto, de dos puntos, doble y uniforme. Para todos estos cruzamientos, se toman parejas de individuos y, con una probabilidad, se decide si se emparejan ó no, de acuerdo al tipo de emparejamiento que se está realizando. Si se decide no emparejar dos individuos, éstos pasan directamente a formar parte del conjunto de hijos.

A continuación describiremos, en forma resumida, el proceso de los métodos de emparejamiento que vamos a emplear en la evaluación:

- En el emparejamiento simple de un punto, es seleccionado un punto aleatoriamente a través de la longitud del cromosoma y a partir de él se efectúa el cambio del material genético.
- Para el emparejamiento de dos puntos se sigue el mismo procedimiento del emparejamiento de un punto, pero en lugar de escoger un punto son seleccionados dos puntos aleatorios en los cromosomas.
- El emparejamiento doble es básicamente el mismo proceso que el de emparejamiento simple de un punto, excepto que es repetido; esto significa que dos bloques son elegidos aleatoriamente y dos cruzamientos son llevados a cabo.
- El último método utilizado fue el de emparejamiento uniforme. En este método, cada gen del individuo descendiente tiene la misma probabilidad de provenir de alguno de los dos individuos padres, esto significa que cada gen es elegido aleatoriamente de un gen padre.

Dado que inicialmente restringimos un alfabeto para cada gen del cromosoma, el emparejamiento no generará soluciones fuera de este dominio; debido a que los operadores considerados no cambian la posición de los genes, sino que recombinan éstos manteniendo la posición original que tenían en los padres.

Tras realizar las pruebas con los métodos mencionados se llegó a la conclusión: Si los métodos son comparados según el número de

generaciones que se emplea para encontrar una solución factible, y a su vez se considera la frecuencia de éxito en alcanzar dicha solución, entonces se tiene como mejor operador para nuestro problema al Emparejamiento Uniforme (Figura 6.6 y Figura 6.7).

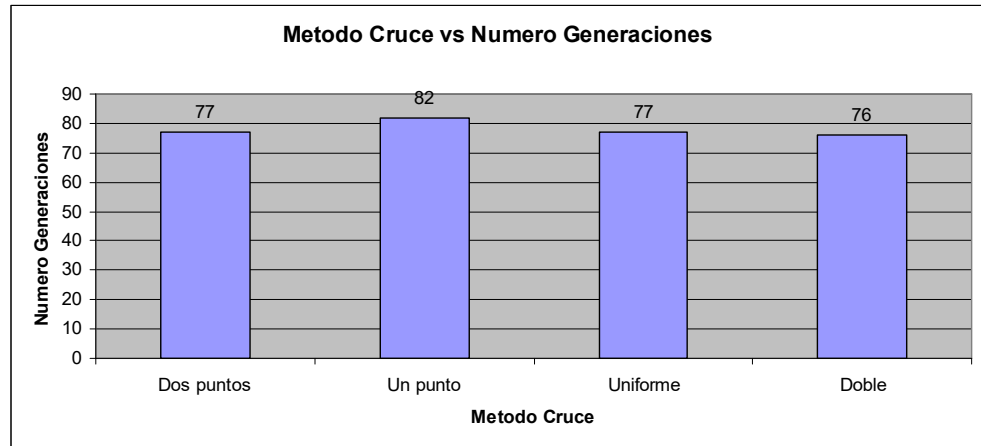


Figura 6.6. Número de generaciones vs. Método de emparejamiento utilizado

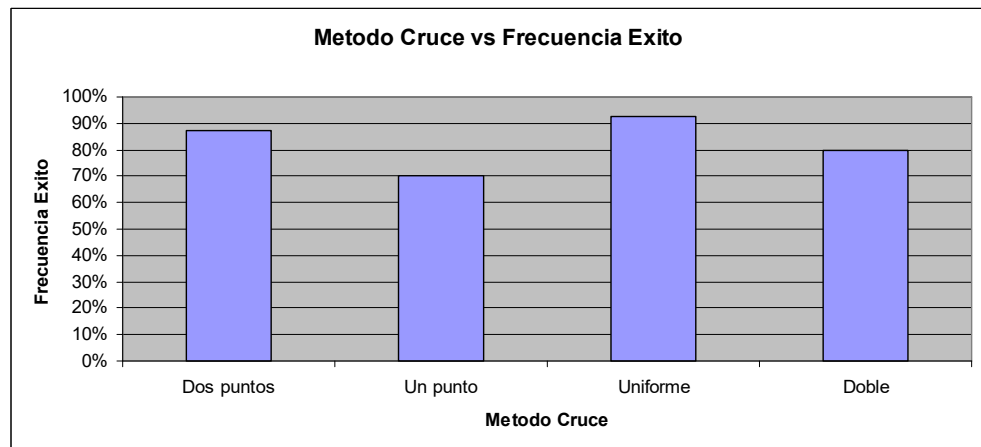


Figura 6.7. Frecuencia de éxito vs. Método de emparejamiento utilizado

Como observación aparte, podemos indicar que el emparejamiento uniforme presenta un poder de destrucción mucho mayor que el emparejamiento de un punto y de dos puntos; esto quiere decir que, el emparejamiento uniforme no conserva la estructura de los cromosomas padres como lo hace el emparejamiento de un punto y dos puntos, que son los que preservan en mayor medida la estructura de los cromosomas padres. Por ejemplo, el individuo hijo

originado a través del emparejamiento uniforme podría estar formado, de manera alternada, por un gen del padre 1 y otro gen del padre 2, con lo cual podemos ver que la estructura original de los cromosomas padres no se ve reflejada en el cromosoma hijo. Pero si se aplicara un emparejamiento de un punto se obtendría un individuo cuya estructura de su cromosoma sería la primera parte del padre 1 y la otra parte del padre 2, con lo cual se puede notar que las estructuras originales de los padres no se ha visto afectada en gran magnitud.

La utilización del emparejamiento uniforme debe ser, por lo tanto, tomada con cuidado en problemas donde la variedad de individuos en la población sea media ó baja; además debe estar presente en un ambiente elitista, que garantice la permanencia de los mejores individuos para futuras generaciones. Precisamente esta descripción se adecua a nuestro problema, dado que el uso de alfabetos que restringen el valor de los genes del cromosoma limita, en cierto modo, la diversidad de individuos dentro de la población.

6.3.2.1.1. Prueba de porcentajes de emparejamiento

El objetivo de esta prueba es conocer el efecto que tiene el operador genético de emparejamiento sobre el funcionamiento del algoritmo, lo cual se logrará modificando el porcentaje de ocurrencia de dicho operador. Los porcentajes con los cuales se realizará la evaluación se muestran en la siguiente tabla:

Porcentaje
50%
60%
70%
80%
90%
100%

Tabla 6.2. Porcentaje de ocurrencia del operador de emparejamiento

En la Figura 6.8 se muestra el comportamiento del algoritmo frente al cambio de porcentajes de emparejamiento. Notándose que el 90% de ocurrencia de emparejamiento logra encontrar, con un menor número de generaciones, la solución al problema.

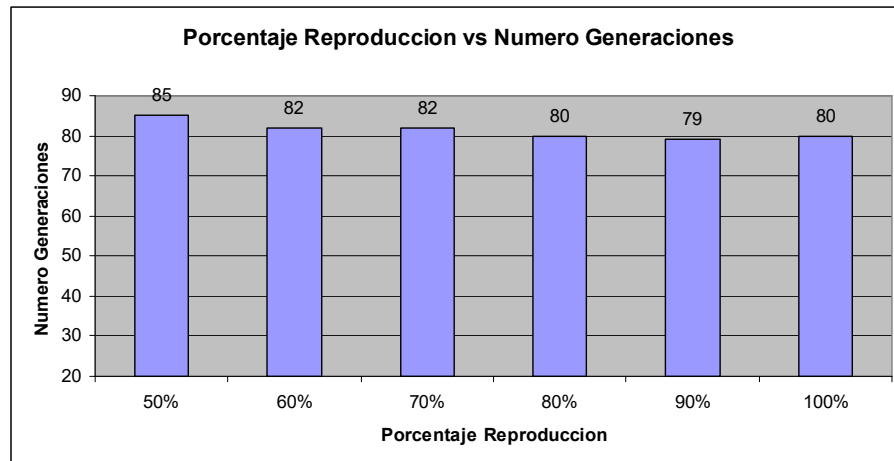


Figura 6.8. Número de generaciones vs. Porcentaje de emparejamiento

6.3.2.2. Prueba sobre el método de mutación

Como se mencionó en el capítulo 4, el operador de mutación ya ha sido seleccionado; debido a que sólo uno se acopla a la manera en que trabaja nuestro algoritmo genético, por lo que se limitará a calcular el porcentaje de mutación adecuado.

6.3.2.2.1. Prueba de porcentaje de mutación

Esta prueba tiene como objetivo medir el efecto de aplicar diferentes porcentajes de mutación sobre el algoritmo genético. Los porcentajes se muestran en la siguiente tabla:

Porcentaje
5%
1%
0.1%
0.01%

Tabla 6.3. Porcentaje de ocurrencia del operador de mutación

Para cada uno de dichos porcentajes se realizaron las pruebas correspondientes, obteniéndose como resultado que el porcentaje de mutación que ofrece encontrar la solución, en un número menor de generaciones, es de 0.1%, seguido del 1% (Figura 6.9). Pero si observamos la Figura 6.10 nos percatamos que el porcentaje de 0.1% sólo posee un 20% de frecuencia de éxito, siendo considerado un valor bajo; por lo cual escogemos como porcentaje de mutación al de 1%, que garantiza una frecuencia mayor al momento de encontrar una solución óptima.

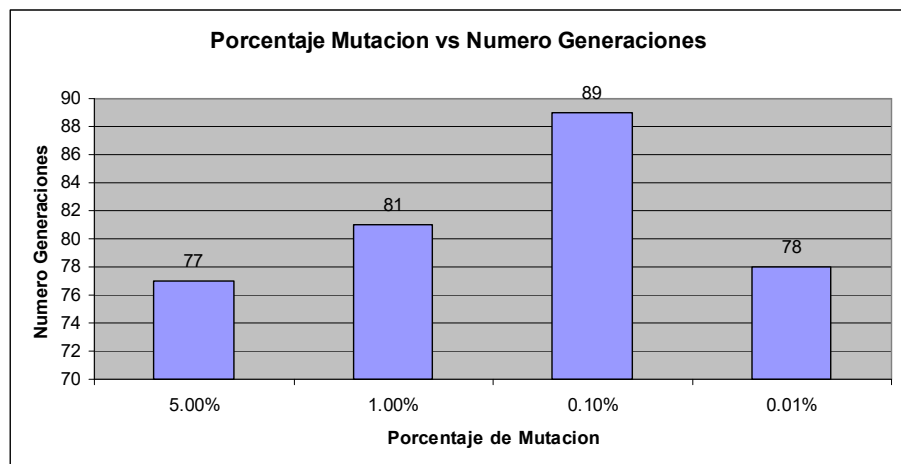


Figura 6.9. Número de generaciones vs. Porcentaje de mutación

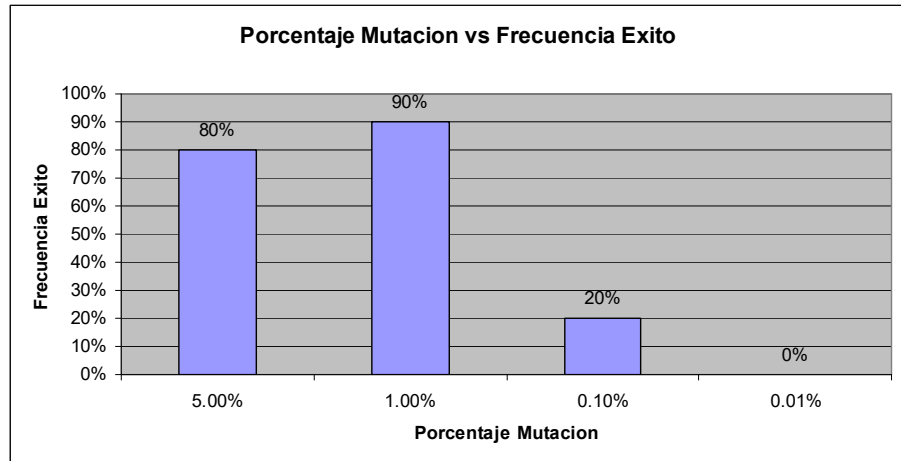


Figura 6.10. Frecuencia de éxito vs. Porcentaje de mutación

6.3.3. Prueba sobre el proceso de reinserción

En esta sección probaremos cuál método de reinserción de los individuos para la siguiente generación es la mejor, así como determinar si el uso del elitismo puede ó no mejorar el desempeño del algoritmo.

6.3.3.1. Prueba sobre el método de reinserción

La operación de reinserción se puede realizar de diversas formas, pero pusimos a prueba sólo los dos métodos que vamos a explicar a continuación:

- La manera más simple es eliminar a toda la población anterior y sustituirla por la nueva población, aquí preservamos solamente los hijos para la siguiente generación.
- El otro método es escoger los mejores cromosomas entre el conjunto de padres e hijos para formar la siguiente generación.

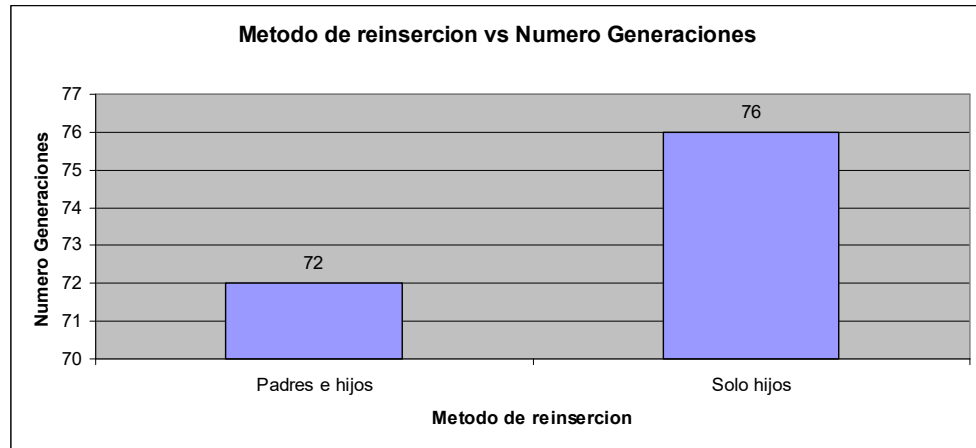


Figura 6.11. Número de generaciones vs. Método de reinserción utilizado

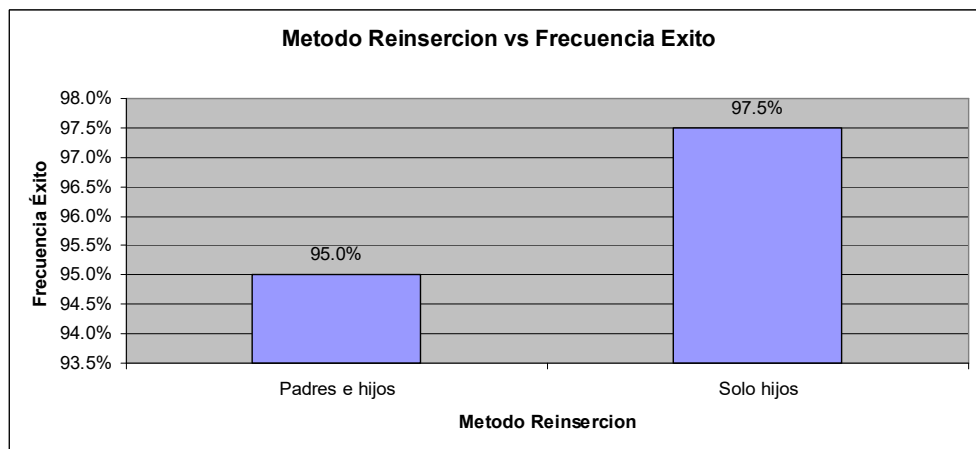


Figura 6.12. Frecuencia de éxito vs. Método de reinserción utilizado

Comparando ambos métodos de reinserción, según las Figuras 6.11 y 6.12, se observa que cuando se consideran los padres e hijos para la reinmersión, el número de generaciones disminuye y además tiene una frecuencia de éxito menor al método que toma en cuenta sólo los hijos; por lo tanto, escogemos el primer método que resulta ser más fiable para encontrar una solución factible.

6.3.3.2. Prueba sobre el uso del elitismo

En este experimento se evaluará si se debe utilizar ó no el elitismo, para determinar si al usarlo se obtienen mejores resultados con respecto a no hacer uso de él. Para este caso consideraremos

mantener únicamente la mejor solución de la población actual para trasladarla a la siguiente generación, siendo solamente dicha solución la que se preserve antes de que se realice el proceso de emparejamiento. Con esto podemos asegurar que si los operadores genéticos no logran generar cromosomas con aptitudes superiores a las aptitudes de la población actual, por lo menos se tendrá la seguridad de conservar el cromosoma con la mejor aptitud que pueda existir en dicha población.

A partir de las pruebas podemos deducir que al preservar la mejor solución para la siguiente generación, el número de generaciones empleado para encontrar una solución óptima al problema es menor que si no la preservamos (Figura 6.13).

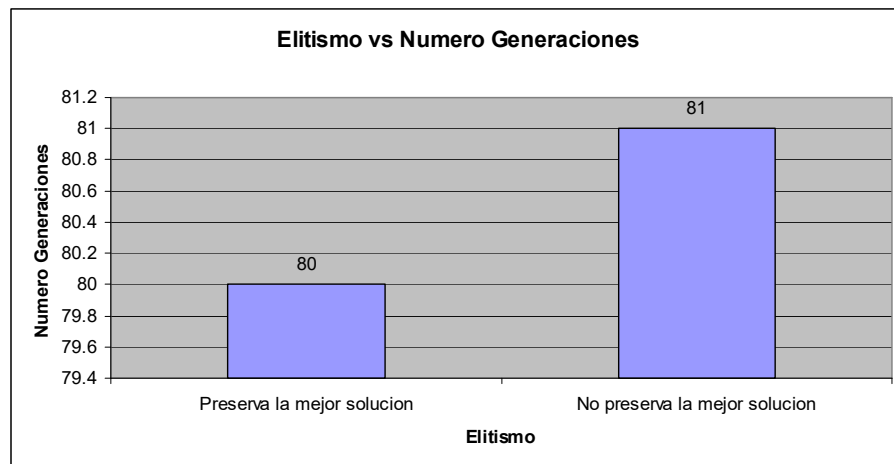


Figura 6.13. Número de generaciones vs. Elitismo

6.3.4. Prueba de parámetros de configuración

En esta sección evaluaremos los valores que deben tomar algunos parámetros de configuración, tales como el número de individuos que formarán la población y el número de generaciones que debe ejecutar el algoritmo. De esta evaluación se determinará cuál es el mejor valor para cada uno de dichos parámetros de configuración, y de esa manera poder encontrar una solución factible al problema.

6.3.4.1. Prueba de número de individuos

El objetivo de esta prueba es encontrar el número de individuos adecuado para cada una de las fases de nuestro algoritmo, con el fin de evitar que éste converja rápidamente en un óptimo local; ya sea por tener una población demasiado pequeña que no aporte mucha diversidad a la solución, ó tener una demasiado grande que consuma muchos recursos computacionales que hacen que el algoritmo avance lentamente.

La Figura 6.14 muestra que para la fase uno el mejor número de individuos de la población inicial, para aportar una buena diversidad, es de 3200, ya que la diversidad de la población no es muy grande ni muy pequeña. La frecuencia de éxito, para hallar una solución óptima, de ese tamaño de población es la más adecuada, ya que partir de este tamaño de población esta no varía en mejorar la frecuencia.

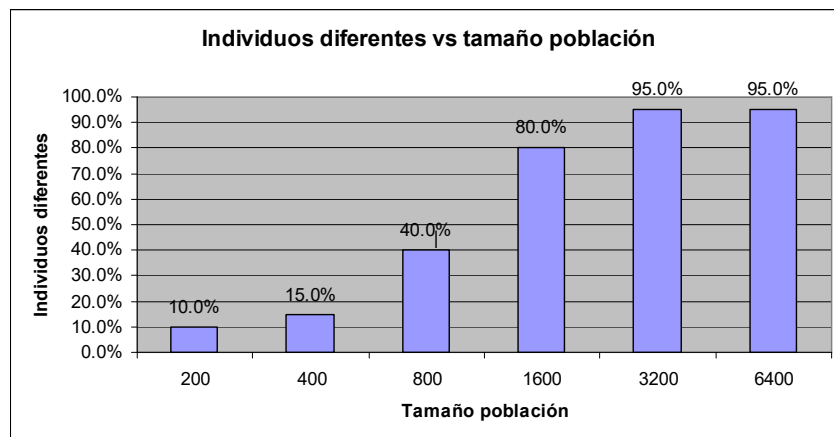


Figura 6.14. Frecuencia de éxito vs. Tamaño de la población inicial - Fase uno

Para la fase dos, se puede notar que el número de individuos de la población es mayor que la fase anterior (6400 individuos). Esto es debido a que se necesita una mayor diversidad de la población para encontrar una solución porque, como sabemos, presenta un mayor número de restricciones. En la Figura 6.15 se muestra la frecuencia de éxito que posee cada uno dichos tamaños.

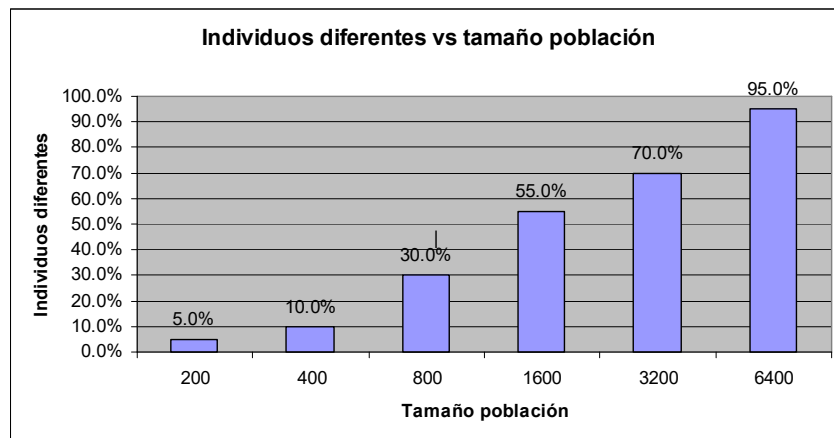


Figura 6.15. Frecuencia de éxito vs. Tamaño de la población inicial - Fase dos

Para la fase tres, el número de individuos de la población es intermedio entre las dos fases anteriores, es decir, mayor que la fase uno y menor que la fase dos. (Figura 6.16)

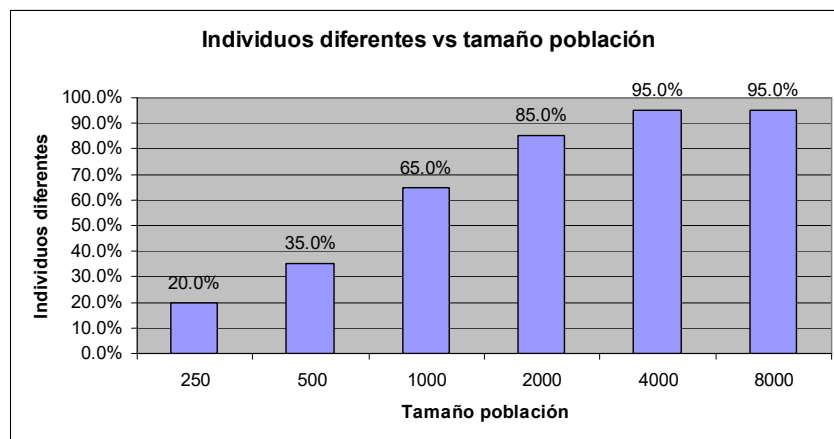


Figura 6.16. Frecuencia de éxito vs. Tamaño de la población inicial - Fase tres

6.3.4.2. Prueba de número de generaciones

En este punto determinaremos el número de generaciones promedio que tardan el algoritmo en cada una de las fases para encontrar una solución. De esta manera se puede averiguar cuál será el máximo número de generaciones adecuado para cada una de dichas fases.

Para realizar esta prueba se empleó un número elevado de generaciones como límite, con la finalidad de visualizar el comportamiento del algoritmo al tratar de encontrar ó no una solución óptima ó de converger prematuramente en un óptimo local. Los resultados obtenidos para cada una de las fases se muestran a continuación:

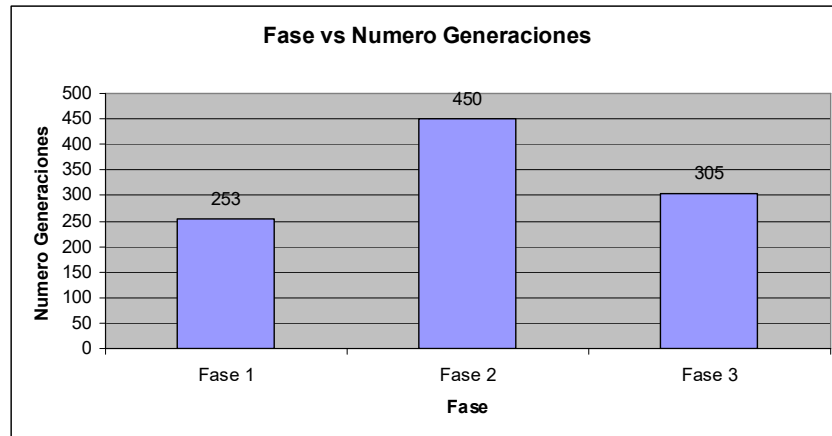


Figura 6.17. Número de generaciones vs. Fase

Como vemos en la Figura 6.20 la fase dos es la que necesita un mayor número de generaciones para encontrar una solución, debido a su elevada complejidad, mientras que la fase tres es relegada al segundo lugar y la primera al último.

6.4. Horario de Clases

Como resultado de la información de entrada, expresada en el caso de análisis número tres, se logró generar el horario de clases que se muestra a continuación:

ALGORITMICA I						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	VI	18	2	HERRERA PONTE, ARON ELÍAS	106
1	P	LU	16	2	HERRERA PONTE, ARON ELÍAS	108
1	L	JU	11	2	VERA POMALAZA, Virginia	306
2	T	JU	12	2	DAMASO RIOS, María Rosa	107
2	P	LU	14	2	HERRERA PONTE, ARON ELÍAS	105
2	L	VI	14	2	VÁSQUEZ PARAGULLA, JULIO	307

3	T	MI	9	2	VÁSQUEZ PARAGULLA, JULIO	107
3	P	SA	18	2	VIVANCO MUÑOZ, PERCY ELÍAS	107
3	L	JU	12	2	MUÑOZ CASILDO, Nehil Indalecio	307
4	T	LU	18	2	MUÑOZ CASILDO, Nehil Indalecio	103
4	P	JU	14	2	SALINAS AZAÑA, Gilberto Aníbal	108
4	L	MI	11	2	MOTA ALVA, Lázaro Flori	306
5	T	MA	16	2	MUÑOZ CASILDO, Nehil Indalecio	108
5	P	JU	16	2	MUÑOZ CASILDO, Nehil Indalecio	103
5	L	SA	17	2	RUIZ DE LA CRUZ MELO, Carlos Augusto	306
6	T	SA	11	2	RUIZ DE LA CRUZ MELO, Carlos Augusto	207
6	P	VI	14	2	VIVANCO MUÑOZ, PERCY ELÍAS	109
6	L	LU	18	2	CUBA MORA, JULIO	306

CALCULO I						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	VI	14	3	VALLALODIT FACIO BENITO	202
1	P	JU	18	2	CARNAHUACA TICSE GREGORIO	203
2	T	MI	9	3	VALLALODIT FACIO BENITO	204
2	P	VI	10	2	VALLALODIT FACIO BENITO	105
3	T	VI	15	3	CARNAHUACA TICSE GREGORIO	205
3	P	MA	15	2	VALLALODIT FACIO BENITO	205
4	T	MA	12	3	VALLALODIT FACIO BENITO	109
4	P	SA	13	2	VALLALODIT FACIO BENITO	106
5	T	SA	14	3	CARNAHUACA TICSE GREGORIO	202
5	P	JU	9	2	VALLALODIT FACIO BENITO	207

MATEMATICA BASICA I						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	LU	8	3	ROJAS CAUTI SAÚL	103
1	P	SA	12	2	ROJAS CAUTI SAÚL	102
2	T	LU	11	3	ROJAS CAUTI SAÚL	206
2	P	MA	15	2	ROJAS CAUTI SAÚL	206
3	T	MI	16	3	ROJAS CAUTI SAÚL	203
3	P	VI	13	2	ROJAS CAUTI SAÚL	201
4	T	JU	18	3	ROJAS CAUTI SAÚL	208
4	P	MI	9	2	ROJAS CAUTI SAÚL	101
5	T	MI	11	3	ROJAS CAUTI SAÚL	107
5	P	MA	18	2	ROJAS CAUTI SAÚL	206

COMPUTACION E INFORMATICA						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	JU	8	3	CASTRO LEÓN, Gloria Helena	203
1	L	SA	15	2	TRUJILLO TREJO, John Ledgard	307
2	T	VI	17	3	TRUJILLO TREJO, John Ledgard	105
2	L	LU	18	2	CASTRO LEÓN, Gloria Helena	301
3	T	MA	18	3	CASTRO LEÓN, Gloria Helena	108

3	L	SA	9	2	DAMASO RIOS, María Rosa	305
4	T	MI	16	3	DAMASO RIOS, María Rosa	202
4	L	MA	17	2	TRUJILLO TREJO, John Ledgard	303
5	T	MI	18	3	TRUJILLO TREJO, John Ledgard	208
5	L	JU	11	2	CASTRO LEÓN, Gloria Helena	302

TEORIA DE SISTEMAS						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	LU	12	3	MOLINA NEYRA, César Alberto	208
2	T	SA	10	3	MOLINA NEYRA, César Alberto	208
3	T	MA	9	3	MOLINA NEYRA, César Alberto	209
4	T	LU	8	3	MOLINA NEYRA, César Alberto	102
5	T	MA	12	3	MOLINA NEYRA, César Alberto	103

TALLER DE TECNICAS DE ESTUDIO						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	MI	17	3	RIVERA BENAVIDES JOSE	206
2	T	SA	14	3	RIVERA BENAVIDES JOSE	209
3	T	JU	16	3	RIVERA BENAVIDES JOSE	102
4	T	LU	11	3	RIVERA BENAVIDES JOSE	102
5	T	MA	9	3	RIVERA BENAVIDES JOSE	108

ALGORITMICA II						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	VI	15	2	HERRERA PONTE, ARON ELÍAS	107
1	P	JU	10	2	VIVANCO MUÑOZ, PERCY ELÍAS	205
1	L	MI	12	2	VERA POMALAZA, Virginia	301
2	T	LU	11	2	VÁSQUEZ PARAGULLA, JULIO	201
2	P	JU	17	2	RUIZ DE LA CRUZ MELO, Carlos Augusto	204
2	L	SA	11	2	CUBA MORA, JULIO	305
3	T	JU	18	2	SALINAS AZAÑA, Gilberto Aníbal	209
3	P	SA	14	2	CUBA MORA, JULIO	207
3	L	MA	9	2	MUÑOZ CASILDO, Nehil Indalecio	307

CALCULO II						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	MI	9	3	GÁLVEZ PÉREZ HUMBERTO	108
1	P	MA	14	2	GÁLVEZ PÉREZ HUMBERTO	204
2	T	LU	8	3	GÁLVEZ PÉREZ HUMBERTO	206
2	P	MA	12	2	GÁLVEZ PÉREZ HUMBERTO	208
3	T	MI	17	3	GÁLVEZ PÉREZ HUMBERTO	108
3	P	JU	9	2	LUNA VALDEZ JUAN	208

MATEMATICA BASICA II						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	VI	12	3	GAVILÁN GONZALES MARUJA	101
1	P	LU	16	2	GAVILÁN GONZALES MARUJA	207
2	T	MI	9	3	GAVILÁN GONZALES MARUJA	102
2	P	SA	17	2	MARTINEZ ALIAGA YOVER	106
3	T	LU	18	3	GAVILÁN GONZALES MARUJA	108
3	P	VI	14	2	MARTINEZ ALIAGA YOVER	208

FISICA GENERAL						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	MA	10	2	ALARCON VELAZCO PABLO	103
1	P	MI	18	2	CALVO CHIA MARTIN	201
1	L	JU	14	2	CALVO CHIA MARTIN	304
2	T	JU	15	2	ALARCON VELAZCO PABLO	105
2	P	LU	14	2	CALVO CHIA MARTIN	201
2	L	VI	12	2	CALVO CHIA MARTIN	302
3	T	MA	16	2	ALARCON VELAZCO PABLO	203
3	P	SA	16	2	CALVO CHIA MARTIN	204
3	L	MI	11	2	CALVO CHIA MARTIN	303

ESTRUCTURA DE DATOS						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	SA	18	3	HERRERA PONTE, ARON ELÍAS	202
1	P	LU	11	2	SALINAS AZAÑA, Gilberto Aníbal	106
2	T	LU	16	3	SALINAS AZAÑA, Gilberto Aníbal	107
2	P	MA	14	2	DEL PINO RODRÍGUEZ, Luz	107
3	T	SA	9	3	CORTEZ VASQUEZ, Augusto	107
3	P	JU	15	2	DEL PINO RODRÍGUEZ, Luz	205

ECONOMIA						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	JU	18	2	PALOMINO LADRÓN MARGARITA	106
2	T	VI	10	2	REATEGUI SANCHEZ LLEYNI	204

ALGORITMICA III						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	SA	16	2	CHÁVEZ SOTO, Jorge	104
1	P	VI	8	2	CORTEZ VASQUEZ, Augusto	108
1	L	MA	17	2	HERRERA PONTE, ARON ELÍAS	305
2	T	LU	11	2	CORTEZ VASQUEZ, Augusto	108
2	P	VI	13	2	CHÁVEZ SOTO, Jorge	206
2	L	SA	17	2	CORTEZ VASQUEZ, Augusto	303
3	T	MI	12	2	CORTEZ VASQUEZ, Augusto	102
3	P	SA	14	2	CHÁVEZ SOTO, Jorge	204
3	L	LU	17	2	CHÁVEZ SOTO, Jorge	302

MATEMATICA DISCRETA						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	MI	18	3	QUINTO PAZCE, Daniel Alfonso	205
1	P	VI	18	2	QUINTO PAZCE, Daniel Alfonso	206
2	T	MI	15	3	QUINTO PAZCE, Daniel Alfonso	103
2	P	JU	16	2	QUINTO PAZCE, Daniel Alfonso	208

ESTADISTICA I						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	JU	18	3	ZACARIAS DIAZ MARIA	103
1	P	LU	15	2	ZACARIAS DIAZ MARIA	101
2	T	LU	18	3	ZACARIAS DIAZ MARIA	209
2	P	MA	14	2	VEGA RENGIFO LIDA	209
3	T	JU	14	3	ZACARIAS DIAZ MARIA	107
3	P	SA	16	2	ZACARIAS DIAZ MARIA	109

DISEÑO GRAFICO						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	LU	18	2	TRUJILLO TREJO, John Ledgard	208
1	L	SA	12	2	SOLARI CARBAJAL, GABRIEL	302
2	T	SA	10	2	SOLARI CARBAJAL, GABRIEL	103
2	L	JU	14	2	TRUJILLO TREJO, John Ledgard	307
3	T	SA	8	2	SOLARI CARBAJAL, GABRIEL	103
3	L	JU	17	2	TRUJILLO TREJO, John Ledgard	303

ELECTROMAGNETISMO						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	MI	12	2	SALAZAR DE PAZ LUIS	208
1	L	MA	13	2	VILCAPOMA LAZARO LUIS	305
2	T	MA	17	2	VILCAPOMA LAZARO LUIS	208
2	L	JU	11	2	SALAZAR DE PAZ LUIS	303

ORGANIZACIÓN Y ADMINISTRACION						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	JU	15	3	DE LA CRUZ VÉLEZ DE VILLA, Percy	101
2	T	MI	9	3	DE LA CRUZ VÉLEZ DE VILLA, Percy	203
3	T	MA	8	3	DE LA CRUZ VÉLEZ DE VILLA, Percy	203

TECNICAS DE MODELAMIENTO						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	JU	15	3	LUZA MONTERO, César	106
1	L	MA	18	2	LUZA MONTERO, César	307
2	T	JU	18	3	LUZA MONTERO, César	201
2	L	VI	10	2	ALCÁNTARA LOAYZA, César Augusto	302

COMPUTACION GRAFICA						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	LU	11	3	BUSTAMANTE OLIVERA, Víctor Hugo	107
1	L	SA	10	2	BUSTAMANTE OLIVERA, Víctor Hugo	304
2	T	MA	9	3	BUSTAMANTE OLIVERA, Víctor Hugo	106
2	L	JU	14	2	BUSTAMANTE OLIVERA, Víctor Hugo	306
3	T	JU	17	3	BUSTAMANTE OLIVERA, Víctor Hugo	205
3	L	LU	18	2	BUSTAMANTE OLIVERA, Víctor Hugo	305

ECUACIONES DIFERENCIALES						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	MA	8	3	ACUÑA MONTAÑEZ WALTER	208
1	P	VI	12	2	VILLANUEVA YAYA AMELIA	203
2	T	MA	16	3	ACUÑA MONTAÑEZ WALTER	102
2	P	VI	18	2	VILLANUEVA YAYA AMELIA	107
3	T	VI	9	3	VILLANUEVA YAYA AMELIA	109
3	P	SA	8	2	VILLANUEVA YAYA AMELIA	208

ESTADISTICA II						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	VI	9	3	MIRANDA ROBLES MARY	106
1	P	MI	15	2	MIRANDA ROBLES MARY	108
2	T	SA	11	3	CANALES DEL MAR MANUEL	206
2	P	JU	9	2	MIRANDA ROBLES MARY	108

CIRCUITOS DIGITALES						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	MI	12	2	FERMÍN PÉREZ, Félix Armando	204
1	P	LU	9	2	FERMÍN PÉREZ, Félix Armando	104
1	L	JU	9	2	SUMOSO HUAMAN, ANGEL ORLANDO	303
2	T	MA	14	2	MORI TORRES, SANTIAGO GARDIL	202
2	P	MI	15	2	MORI TORRES, SANTIAGO GARDIL	102
2	L	VI	14	2	SUMOSO HUAMAN, ANGEL ORLANDO	306

CONTABILIDAD GENERAL						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	MI	18	2	ROJAS BORJAS	101
2	T	LU	9	2	ROJAS BORJAS	205

BASE DE DATOS						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	SA	13	3	GALINDO HUAYLLANI, José Luis	205
1	L	MI	12	2	CHÁVEZ SOTO, Jorge	307
2	T	MI	15	3	DEL PINO RODRÍGUEZ, Luz	101
2	L	SA	9	2	LÓPEZ VILLANUEVA, Pablo Edwin	306

3	T	LU	11	3	CHÁVEZ SOTO, Jorge	203
3	L	SA	11	2	GALINDO HUAYLLANI, José Luis	307

INVESTIGACION OPERATIVA						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	LU	18	3	VALVERDE AYALA, Melva	201
1	P	MI	15	2	DELGADILLO AVILA, Rosa Sumactika	104
2	T	SA	18	3	CHÁVEZ SOTO, Jorge	205
2	P	VI	11	2	CHÁVEZ SOTO, Jorge	104

METODOS NUMERICOS						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	VI	8	2	QUINTO PAZCE, Daniel Alfonso	103
1	P	JU	14	2	QUINTO PAZCE, Daniel Alfonso	109
2	T	VI	16	2	QUINTO PAZCE, Daniel Alfonso	201
2	P	MI	12	2	QUINTO PAZCE, Daniel Alfonso	203

SISTEMAS DIGITALES						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	VI	18	2	MORI TORRES, SANTIAGO GARDIL	103
1	P	JU	17	2	SUMOSO HUAMAN ANGEL ORLANDO	107
1	L	MA	18	2	MORI TORRES, SANTIAGO GARDIL	306
2	T	MI	18	2	MORI TORRES, SANTIAGO GARDIL	207
2	P	LU	16	2	SUMOSO HUAMAN, ANGEL ORLANDO	203
2	L	MA	15	2	SUMOSO HUAMAN, ANGEL ORLANDO	302

LENGUAJE Y TRADUCTORES						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	VI	11	2	CORTEZ VASQUEZ, Augusto	208
1	P	MA	16	2	CORTEZ VASQUEZ, Augusto	202
2	T	VI	14	2	RUIZ DE LA CRUZ MELO, Carlos Augusto	105
2	P	JU	13	2	CORTEZ VASQUEZ, Augusto	207
3	T	SA	13	2	RUIZ DE LA CRUZ MELO, Carlos Augusto	104
3	P	JU	14	2	RUIZ DE LA CRUZ MELO, Carlos Augusto	203

CONTABILIDAD GERENCIAL						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	JU	9	2	OSPINO EDERY JOSE	107
1	P	MI	9	2	OSPINO EDERY JOSE	105
2	T	SA	16	2	OSPINO EDERY JOSE	205
2	P	MA	10	2	OSPINO EDERY JOSE	102

3	T	MI	18	2	OSPINO EDERY JOSE	209
3	P	MA	15	2	OSPINO EDERY JOSE	104

SISTEMAS OPERATIVOS						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	MA	19	3	CORAL YGNACIO, Marco Antonio	104
1	L	VI	9	2	DIAZ MUÑANTE, Jorge Raúl	304
2	T	VI	16	3	DIAZ MUÑANTE, Jorge Raúl	104
2	L	LU	12	2	DIAZ MUÑANTE, Jorge Raúl	307

ANALISIS DE SISTEMAS						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	JU	11	3	CORAL YGNACIO, Marco Antonio	201
1	P	MI	15	2	CORAL YGNACIO, Marco Antonio	107
2	T	JU	14	3	CORAL YGNACIO, Marco Antonio	204
2	P	SA	13	2	CORAL YGNACIO, Marco Antonio	201

INVESTIGACION OPERATIVA II						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	MI	8	2	VALVERDE AYALA, Melva	205
1	P	MA	9	2	VALVERDE AYALA, Melva	109
2	T	VI	9	2	VALVERDE AYALA, Melva	101
2	P	MA	17	2	VALVERDE AYALA, Melva	106
3	T	MI	21	2	SOLARI CARBAJAL, GABRIEL	209
3	P	JU	10	2	VALVERDE AYALA, Melva	102

TRANSMISION DE DATOS						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	MI	21	2	MOLINA NEYRA, César Alberto	208
1	P	VI	14	2	CARRASCO OR?, Nilo Eloy	209
1	L	MA	15	2	CARRASCO OR?, Nilo Eloy	303
2	T	JU	19	2	ARMAS CALDERÓN, Raúl	204
2	P	SA	11	2	ARMAS CALDERÓN, Raúl	204
2	L	MA	21	2	ARMAS CALDERÓN, Raúl	306

ARQUITECTURA DE COMPUTADORAS						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	MI	13	2	GONZALES SUAREZ, Juan Carlos	106
1	P	VI	21	2	MAC DOWALL REINOS, Erwin	106
2	T	SA	9	2	GONZALES SUAREZ, Juan Carlos	203
2	P	MI	16	2	CONTRERAS FLORES, Walter Pedro	105

TALLER DE PROYECTOS						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	SA	13	2	MAMANI RODRÍGUEZ, Zoraida Emperatriz	109
1	P	JU	19	2	MAMANI RODRÍGUEZ, Zoraida Emperatriz	102
2	T	VI	14	2	CORAL YGNACIO, Marco Antonio	204
2	P	SA	18	2	ALARCÓN LOAYZA, Luis	208
3	T	LU	13	2	VEGA HUERTA, Hugo Froilan	204
3	P	VI	20	2	LA SERNA PALOMINO, Nora	105

Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	JU	19	3	MAC DOWALL REINOS, Erwin	101
1	L	SA	13	2	MOLINA NEYRA, Cesar Alberto	304
2	T	SA	20	3	MAC DOWALL REINOS, Erwin	203
2	L	VI	19	2	MAC DOWALL REINOS, Erwin	306

DISEÑO DE SISTEMAS						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	VI	12	3	ALCÁNTARA LOAYZA, César Augusto	108
1	P	LU	12	2	YÁÑEZ DURAN, Carlos Enrique	103
2	T	VI	10	3	YÁÑEZ DURAN, Carlos Enrique	102
2	P	SA	9	2	CORAL YGNACIO, Marco Antonio	202

DISEÑO DE INTERFACES DE USUARIO						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	LU	9	3	DÍAZ MUÑANTE, Jorge Raúl	105
1	P	MI	8	2	DÍAZ MUÑANTE, Jorge Raúl	109
2	T	JU	12	3	YÁÑEZ DURAN, Carlos Enrique	106
2	P	MI	11	2	YÁÑEZ DURAN, Carlos Enrique	205

INTELIGENCIA ARTIFICIAL						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	MA	11	3	VERA POMALAZA, Virginia	107
1	L	JU	15	2	MAURICIO SÁNCHEZ, David Santos	302
2	T	VI	15	3	MAURICIO SÁNCHEZ, David Santos	206
2	L	JU	9	2	VEGA HUERTA, Hugo Froilan	305

MODELOS Y SIMULACION						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	MI	20	2	RIVAS PEÑA, Marcos	102
1	P	LU	19	2	SOLARI CARBAJAL, GABRIEL	204
1	L	SA	8	2	RIVAS PEÑA, Marcos	301
2	T	MI	19	2	SOLARI CARBAJAL, GABRIEL	109
2	P	VI	21	2	SOLARI CARBAJAL, GABRIEL	204
2	L	JU	21	2	SOLARI CARBAJAL, GABRIEL	304

INGENIERIA DE INFORMACION						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	LU	8	3	PIEDRA ISUSQUI, José	203
1	L	MI	16	2	PIEDRA ISUSQUI, José	302
2	T	SA	13	3	LÓPEZ VILLANUEVA, Pablo Edwin	101
2	L	MI	20	2	PIEDRA ISUSQUI, José	305

SISTEMAS DISTRIBUIDOS						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	JU	16	2	GUERRA GUERRA, Jorge	203
1	P	VI	14	2	GUERRA GUERRA, Jorge	104
1	L	MI	8	2	GUERRA GUERRA, Jorge	305
2	T	JU	14	2	GUERRA GUERRA, Jorge	201
2	P	SA	11	2	GUERRA GUERRA, Jorge	101
2	L	MA	13	2	GUERRA GUERRA, Jorge	302

INGENIERIA DE SOFTWARE						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	SA	19	2	LÓPEZ VILLANUEVA, Pablo Edwin	103
1	L	MA	21	2	LÓPEZ VILLANUEVA, Pablo Edwin	302
2	T	VI	10	2	BUSTAMANTE OLIVERA, Víctor Hugo	201
2	L	LU	20	2	LÓPEZ VILLANUEVA, Pablo Edwin	306
3	T	SA	17	2	LÓPEZ VILLANUEVA, Pablo Edwin	103
3	L	MI	18	2	BUSTAMANTE OLIVERA, Víctor Hugo	303

METODOLOGIA PARA LA ELABORACION DE TESIS						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	MI	11	3	ROMÁN CONCHA, Norberto Ulises	206
1	L	JU	13	2	ROMÁN CONCHA, Norberto Ulises	305

TALLER DE PROYECTOS						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	JU	19	3	LAM, Zhing Fong	207
1	L	VI	21	2	MAMANI RODRÍGUEZ, Zoraida Emperatriz	304

INGENIERIA DE CONTROL						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	MA	11	2	FERMÍN PÉREZ, Félix Armando	104
1	P	MI	18	2	FERMÍN PÉREZ, Félix Armando	204
2	T	VI	13	2	FERMÍN PÉREZ, Félix Armando	207
2	P	MA	9	2	FERMÍN PÉREZ, Félix Armando	202

DISEÑO DE UNA ARQUITECTURA WEB						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	P	SA	14	2	GUERRA GUERRA, Jorge	107
1	L	MI	21	2	GUERRA GUERRA, Jorge	304

PLANAMIENTO DE RECURSOS EMPRESARIALES						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	SA	8	2	GALINDO HUAYLLANI, José? Luis	109

DATAWAREHOUSE						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	P	VI	19	2	PANTOJA COLLANTES, Jorge Santiago	202
1	L	MA	19	2	PANTOJA COLLANTES, Jorge Santiago	303

AUDITORIA Y SEGURIDAD INFORMATICA						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	LU	15	3	PANTOJA COLLANTES, Jorge Santiago	202
2	T	MI	19	3	PANTOJA COLLANTES, Jorge Santiago	202

INGENIERIA DE NEGOCIOS						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	JU	19	2	GALINDO HUAYLLANI, José Luis	108
2	T	MA	19	2	GALINDO HUAYLLANI, José Luis	103

MARKETING						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	SA	20	2	VEGA HUERTA, Hugo Froilan	107
2	T	LU	21	2	VEGA HUERTA, Hugo Froilan	104
3	T	MI	19	2	VEGA HUERTA, Hugo Froilan	107

DERECHO INFORMATICO						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	VI	21	2	MOLINA SALAS, ROMEL	202
2	T	JU	20	2	MOLINA SALAS, ROMEL	109

PROYECTO DE TESIS I						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	MI	20	3	GIL CALVO, Rubén	204

FORMULACION Y EVALUACION DE PROYECTOS						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	JU	17	2	ORTEGA LOAYZA, DANIEL	207
1	P	VI	19	2	OSORIO BELTRÁN, Norberto Antonio	109

2	T	MI	15	2	OSORIO BELTRÁN, Norberto Antonio	201
2	P	VI	15	2	OSORIO BELTRÁN, Norberto Antonio	103

SISTEMAS INTELIGENTES						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	MI	16	2	MAGUIÑA PÉREZ, Rolando Alberto	209

COMERCIO ELECTRONICO						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	SA	16	3	GALINDO HUAYLLANI, José? Luis	108

REDES NEURONALES						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	LU	18	2	MAGUIÑA PÉREZ, Rolando Alberto	102

GERENCIA INFORMATICA						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	SA	14	3	PIEDRA ISUSQUI, José	103
2	T	SA	17	3	PIEDRA ISUSQUI, José	105

TENDENCIAS TECNOLOGICAS						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	LU	17	2	BUSTOS DÍAZ, Silverio	106
1	P	JU	19	2	BUSTOS DÍAZ, Silverio	105
2	T	JU	15	2	BUSTOS DÍAZ, Silverio	206
2	P	MI	17	2	BUSTOS DÍAZ, Silverio	106

GESTION DE PROYECTOS INFORMATICOS						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	VI	21	2	LEÓN FERNÁNDEZ, Cayo Víctor	206
1	P	JU	17	2	PÉREZ QUINTANILLA, José? Antonio	206
2	T	LU	18	2	OSORIO BELTRÁN, Norberto Antonio	206
2	P	VI	18	2	PÉREZ QUINTANILLA, José? Antonio	205

PROYECTO DE TESIS II						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	MA	17	3	MAURICIO SÁNCHEZ, David Santos	204
2	T	MA	20	3	LUZA MONTERO, César	202

SEMINARIO DE LA REALIDAD PERUANA						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	LU	20	2	MOLINA SALAS, ROMEL	203
2	T	MI	19	2	MOLINA SALAS, ROMEL	105

GESTION DE CONOCIMIENTOS						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	MI	16	2	VERA POMALAZA, Virginia	109

LIDERAZGO EMPRESARIAL						
Grupo	Tipo	Día	Hora	Dur.	Docente	Aula
1	T	LU	15	2	ENRIQUEZ MAGUIÑA, Wiliam Mart	204

Capítulo 7 : Conclusiones y Recomendaciones

En esta sección detallaremos las conclusiones que se derivaron de este trabajo, así como las respectivas recomendaciones para posibles trabajos futuros.

7.1. Conclusiones

La Facultad de Ingeniería de Sistemas e Informática de la UNMSM no utiliza herramientas computacionales para generar los horarios de clase, los cuales manejan un alto grado de complejidad durante su elaboración. Es por esto que se planteó diseñar una herramienta basada en algoritmos genéticos, que permita cubrir las necesidades existentes y en cuyo proceso de investigación se llegó a las conclusiones que se presentan a continuación:

1. Se construyó un algoritmo que automatiza el proceso puntual de la generación de horarios de clases y que permite encontrar soluciones buenas y/o consideradas aceptables dentro del margen de error definido y las restricciones planteadas. Con lo cual se deduce que el objetivo general de esta investigación, que es encontrar una solución mediante el uso de algoritmos genéticos al

problema de asignación de docentes, periodos y aulas a los cursos, se ha cumplido a cabalidad.

2. Otra conclusión, desprendida del proceso de investigación, está relacionada con la división del problema en distintas fases. Inicialmente no se pudo apreciar el impacto total de su aplicación sobre el problema, pero una vez concluida la investigación se observó lo siguiente:

- 2.1. La división por fases permite reducir el tiempo de procesamiento y el uso de recursos; debido a que en la medida que se van encontrando soluciones en cada fase, las cuales resultan ser información de entrada de la fase subsecuente, el espacio de soluciones se va acotando cada vez más. La aplicación de la técnica “divide y vencerás” resultó de gran ayuda al momento de tratar de resolver el problema a partir de la solución de subproblemas.

Otro logro conseguido gracias a la división en fases fue la simplicidad obtenida en la implementación del algoritmo, debido a que las restricciones relacionadas a los docentes, periodos y aulas fueron tomadas en cuenta de manera separada unas de otras, de acuerdo al recurso asociado a cada fase.

- 2.2. Se pudo observar una debilidad en el método. Pues tal vez al momento de acotar el espacio de soluciones mediante la generación de una solución parcial por parte de una fase anterior, se perdían ciertas soluciones factibles que ya no podrían ser exploradas por las fases subsecuentes y en la cual podría estar la solución que estamos buscando. Para solucionar esto se planteó maximizar la diversidad de individuos dentro de la población, de tal manera que nuestra población se encuentre lo más distribuido posible dentro del espacio de búsqueda acotado donde el algoritmo pueda desplazarse. Esto significa hacer que los individuos sean lo más diferentes posibles en la población sin interferir con los procesos de evolución normal.

7.2. Recomendaciones

Entre las recomendaciones y mejoras a realizar, una vez terminada la investigación, se tiene lo siguiente:

1. Combinar el algoritmo genético con otra metaheurística para mejorar el tiempo de generación y calidad de la solución. Por ejemplo, la función que se encarga de calcular la aptitud de cada individuo de la población en las diferentes fases podría ser remplazada por una red neuronal, la cual se encargaría de realizar dicho cálculo. Con lo cual se dotaría a este nuevo algoritmo híbrido una característica de aprendizaje adicional a una búsqueda guiada.

A su vez podría utilizarse una heurística de búsqueda, tal como un algoritmo voraz, que se encargue de generar la población inicial con cierto grado de pre-procesamiento y que por ende sea mejor que simplemente inicializar la población de manera aleatoria.

2. La segunda recomendación es implementar el sistema según el diseño planteado en el presente trabajo e implantarlo en la Facultad de Ingeniería de Sistemas e Informática de la UNMSM para su respectivo uso.
3. Implementar un servicio Web con este algoritmo, de tal manera que no sea usado solamente por la facultad sino también abrir la posibilidad de brindar este servicio a otras facultades y/o instituciones que lo requieran. Otra ventaja de implementar el servicio sería que se estaría destinando la carga del procesamiento a un servidor dedicado para liberar al cliente de dicha tarea.
4. Una vez establecido el procesamiento del algoritmo en un servidor, podrá ser organizado sobre una arquitectura de procesamiento distribuida. En este punto se presentan varias opciones, una es armar una arquitectura de computación paralela (computación grid) ya que proporciona una gran cantidad de recursos según se necesite, claro que este modelo sería independiente al algoritmo y tendría que realizarse un trabajo previo para implementar esta arquitectura. Otra opción más ligada a la implementación de la solución sería elaborar un modelo de procesamiento maestro-esclavo, donde un servidor maestro controle el proceso evolutivo, concentre la

información resultante y asigne carga de trabajo a sus servidores esclavos. Dicha carga de trabajo podría ser la evaluación de una parte de los individuos de la población, dado que esto es lo que más tiempo toma en procesar.

5. Otra mejora, ya no tanto basándonos en el algoritmo ni en la arquitectura sobre la cual podría implementarse, estaría relacionada con las características del posible software, de tal manera que pueda ser más configurable y extensible. Extensible, en cuanto a su uso para otras realidades similares y a su vez poder generar horarios no solamente para una facultad sino tal vez para toda la universidad en una sola ejecución del algoritmo. Con la finalidad de hacer que el software sea más configurable, se recomienda permitir a los futuros usuarios poder escoger entre todas las restricciones que maneja el sistema y decidir cuáles se acoplan a su realidad o problema específico.
6. Aumentar el potencial del software al momento de la generación de horarios a través de su integración con la información contenida en el Sistema Único de Matricula de la UNMSM, con la cual se podría calcular de manera automática la cantidad de grupos que se deberían aperturar para los diferentes cursos de manera automática o por lo menos sugerirlos para su aprobación.
7. Normar los tiempos de entrega sobre la disponibilidad laboral de los docentes así como sus preferencias de dictado. Esto permitirá brindar al alumnado horarios de manera oportuna y con información más confiable y precisa.
8. Normar el uso del software que vaya a ser implementado para evitar que factores externos al proceso normal de generación de horarios pudieran invalidar el uso de los horarios generados por dicho sistema.

Bibliografía

Libros

- 1 Dawkins R. (1976). "The Selfish Gene". USA, New York:Oxford University Press
- 2 Díaz, Adenso (1996). Optimización heurística y redes neuronales. España: Editorial Paraninfo
- 3 Fogel, David B (1995). Evolutionary Computation: Toward a New Philosophy of Machine Intelligence. USA :IEEE Press
- 4 Garey, Michael R. (1979). Computers and intractability. A guide to the Theory of NP- Completeness. USA: Bell Telephone Laboratories
- 5 Gil Londoño, Natyhelem (2006). Algoritmos Genéticos. Medellín
- 6 Gil Tallavo, Marcos y Antonio Martínez, Amadís (2006). "Algoritmo basado en Tabú Search para el Problema de Asignación de Horarios de Clases". Departamento de Computación Facultad de Ciencias y Tecnología Universidad de Carabobo Valencia, Estado Carabobo, Venezuela
- 7 Glover, F. y Laguna, M. (1993). Tabú Search. Modern Heuristic Techniques for Combinatorial Problems, USA: C. Reeves, ed. Blackwell Scientific Publishing
- 8 Goldberg, D. y Voessne, S (1999). "Optimizing global-local search hybrids". In GECCO'99: Proceedings of the Genetic and Evolutionary Computation Conference. p 220-228
- 9 Goldberg, David E. (1989). "Genetic Algorithms in Search, Optimization, and Machine Learning", Addison-Wesley Publishing Company
- 10 Johnsonbaugh, Richard (1999). Matemáticas Discretas. Cuarta Edición, Prentice Hall
- 11 Kaufmann, A. y Faure, R. (1977). "Invitación a la Investigación de Operaciones", Segunda Edición, México: CECSA
- 12 Koza, John R. (1992). Genetic Programming. On the Programming of Computers by Means of Natural Selection. USA: The MIT Press.
- 13 Krasnogor, N. y Smith, J. (2005) "A Tutorial for Competent Memetic Algorithms: Model, Taxonomy and Design Issues". IEEE Transactions on Evolutionary Computation.
- 14 Leiserson, C. Rivest, R. y Stein, C. Cormen, T. (2001). Introduction to algorithms. USA: The MIT Press McGrawHill
- 15 López Ceballo, Paulina Danae (2000). "El Método de búsqueda Tabú para la programación de horarios". México: Editorial Universidad de Sonora
- 16 Macías, E. A. (2004). "Algoritmos de Búsqueda Local de GRASP para Solución de 3-SAT. p. 219-225
- 17 Merz, P. y Freisleben, B.(1999) "A comparison of memetic algorithms, tabú search, and ant colonies for the quadratic assignment problem". In Proceedings of the 1999 International Congress of Evolutionary Computation (CEC' 99), Washington DC, USA
- 18 Michalewicz, Zbigniew (1996). "Genetic algorithms + data structures = evolution programs". Alemania: Editorial Springer
- 19 Moscazo, P. (1989). "On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms". California Institute of Technology, Pasadena, California, USA, Tech. Rep. Caltech Concurrent Computation Program, Reporte 826
- 20 Saleh M.A, Elmohamed y Fox, Geoffrey (1998) "A comparison of Annealing Techniques for Academic Course Scheduling". Practice and Theory of Automated Timetabling II, Selected Papers from the 2nd International Conference, PATAT'97
- 21 Vázquez Espi, Mariano(1994) . "Recocido simulado: un nuevo algoritmo para la optimización de estructuras". Madrid
- 22 Whitley, Darrell (1997). A genetic algorithm tutorial. Holanda: Editorial Springer

Revistas

- 23 Larrosa, J. y Meseguer (2003). Restricciones blandas: modelos y algoritmos. Revista Iberoamericana de Inteligencia Artificial, Volumen Otoño/2003, Numero 20, p.69-81
- 24 Alkan, Alpay y Özcan, Ender (2003). "Memetic Algorithms for Timetabling". Evolutionary Computation, 2003. CEC '03. The 2003 Congress on, Volumen 3, p 1796-1802
- 25 Berretta R, Cotta C, y Moscato P. (2003) "Enhancing the performance of memetic algorithms by using a matching-based recombination algorithm: Results on the number partitioning problem". In M. Resende and J. Pinho de Sousa, editors, Metaheuristics: Computer Decision Making. Kluwer Academic Publishers, Boston MA
- 26 Casado Yusta, Silvia y Pacheco Bonrostro, Joaquín (2003). "Estudio Comparativo de diferentes estrategias meta heurísticas para la resolución del Labor Scheduling Problem". Estudios de Economía Aplicada, Volumen Diciembre/2003, Numero 21, p 537-554
- 27 Franco Baquero, John Fredy, Eliana Miredy Toro Ocampo y Ramón Alfonso Gallego Rendón (2008). Course timetabling problem resolved using Tabu Search. Ingeniería y Desarrollo N° 24.
- 28 Glover, F (1989). Tabú search - part I. ORSA Journal on Computing, Volumen Verano/2003, Numero 3
- 29 Kirkpatrick S., Gelatt C.D. y Vecchi M.P, (1983) "Optimization by Simulated Annealing". Science, Volumen 220, Numero 4598
- 30 Melián, B., Moreno Pérez, J.A., Moreno Vega, J.M. (2003): Meta heurísticas: Una visión global. Revista Iberoamericana de Inteligencia Artificial 19, 2, 7-28
- 31 Mushi, A. R. (2007). "Simulated Annealing Algorithm for the Examinations Timetabling Problem". African Journal of Science and Technology (AJST) Science and Engineering Series Volumen. 8, Numero. 2, p. 24 - 32
- 32 Naji Azimi, Zahra (2004). Comparison of metaheuristic algorithms for examination timetabling problem. J. Appl. Math. & Computing Vol. 16(2004), No. 1 - 2, pp. 337 - 354

Fuentes Electrónicas

- 33 Algoritmos genéticos y computación evolutiva. Abril 2004 (Referido 01/10/2009). Disponible en URL: <http://the-geek.org/docs/algen/>
- 34 Algoritmo voraz. Julio 2006 (Referido 01/12/2009). Disponible en URL: http://es.wikipedia.org/wiki/Algoritmo_voraz
- 35 Inteligencia artificial. Octubre 2009 (Referido 06/11/2009). Disponible en URL: http://es.wikipedia.org/wiki/Inteligencia_artificial
- 36 Java Docs. Octubre 2009 (Referido 15/12/2009). Disponible en URL: <http://jgap.sourceforge.net/javadoc/3.4.4/>
- 37 Matriz (matemática). Septiembre 2009 (Referido 01/11/2009). Disponible en URL: [http://es.wikipedia.org/wiki/Matriz_\(matem%C3%A1tica\)](http://es.wikipedia.org/wiki/Matriz_(matem%C3%A1tica))
- 38 Programación Lineal. Octubre 2007 (Referido 01/12/2009). Disponible en URL: http://es.wikipedia.org/wiki/Programaci%C3%B3n_lineal
- 39 Red neuronal artificial. Agosto 2008 (Referido 10/11/2009). Disponible en URL: http://es.wikipedia.org/wiki/Red_neuronal_artificial
- 40 Timetable. Octubre 2007 (Referido 01/10/2009). Disponible en URL: <http://en.wikipedia.org/wiki/Timetable>
- 41 Vector (informática). Diciembre 2008 (Referido 01/11/2009). Disponible en URL: [http://es.wikipedia.org/wiki/Matriz_\(programaci%C3%B3n\)](http://es.wikipedia.org/wiki/Matriz_(programaci%C3%B3n))
- 42 Vuelta Atrás. Febrero 2006 (Referido 20/11/2009). Disponible en URL: http://es.wikipedia.org/wiki/Vuelta_Atr%C3%A1s

Apéndice

Apéndice 1: Definición de términos usados

- Timetabling.- es una lista organizada, por lo general establecida en forma de cuadro, con información sobre una serie de eventos organizados: en particular, el momento en que se ha previsto que estos eventos tendrán lugar.[34]
- Horario.- Tiempo en el cual se puede impartir una clase
- Curso.- Conjunto de temas relacionados que se ofrecen como parte de la currícula de una carrera profesional.
- Docente.- Persona que imparte el dictado de uno o más cursos
- Alumno.- Persona que asiste al dictado de uno o más cursos
- Aula.- Espacio físico donde se lleva a cabo una clase
- Clase.-Combinación entre un curso y un horario
- Programación Lineal.- es un procedimiento o algoritmo matemático mediante el cual se resuelve un problema indeterminado, formulado a través de ecuaciones lineales, optimizando la función objetivo, también lineal. Consiste en optimizar (minimizar o maximizar) una función lineal, que denominaremos función objetivo, de tal forma que las variables de dicha función estén sujetas a una serie de restricciones que expresamos mediante un sistema de inecuaciones lineales.[35]
- Programación entera.- Programación entera es programación lineal con la restricción adicional de que los valores de las variables de decisión sean enteros.

- Matriz.- En programación, una matriz o vector es una zona de almacenamiento contiguo, que contiene una serie de elementos del mismo tipo, los elementos de la matriz.[36]

En matemáticas, una matriz es una tabla de números consistente en cantidades abstractas que pueden sumarse y multiplicarse. Las matrices se utilizan para describir sistemas de ecuaciones lineales, realizar un seguimiento de los coeficientes de una aplicación lineal y registrar los datos que dependen de varios parámetros. [37]

- Redes neuronales artificiales.- Las redes de neuronas artificiales son un paradigma de aprendizaje y procesamiento automático inspirado en la forma en que funciona el sistema nervioso de los animales. Se trata de un sistema de interconexión de neuronas en una red que colabora para producir un estímulo de salida. En inteligencia artificial es frecuente referirse a ellas como redes de neuronas o redes neuronales. [38]
- Inteligencia Artificial.- Se denomina inteligencia artificial a la rama de la ciencia informática dedicada al desarrollo de agentes racionales no vivos.. la inteligencia artificial es la disciplina que se encarga de construir procesos que al ser ejecutados sobre una arquitectura física producen acciones o resultados que maximizan una medida de rendimiento determinada, basándose en la secuencia de entradas percibidas y en el conocimiento almacenado en tal arquitectura.[39]
- Backtracking.- En su forma básica, la idea de backtracking se asemeja a un recorrido en profundidad dentro de un grafo dirigido. El grafo en cuestión suele ser un árbol, o por lo menos no contiene ciclos. Sea cual sea su estructura, existe sólo implícitamente. El objetivo del recorrido es encontrar soluciones para algún problema. Esto se consigue construyendo soluciones parciales a medida que progresa el recorrido; estas soluciones parciales limitan las regiones en las que se puede encontrar una solución completa. El recorrido tiene éxito si, procediendo de esta forma, se puede definir por completo una solución. En este caso el algoritmo puede bien detenerse (si lo único que se necesita es una solución del problema) o bien seguir buscando soluciones alternativas (si deseamos examinarlas todas). Por otra parte, el recorrido no tiene éxito si en alguna etapa la solución parcial construida hasta el momento no se puede completar. En tal caso, el recorrido vuelve

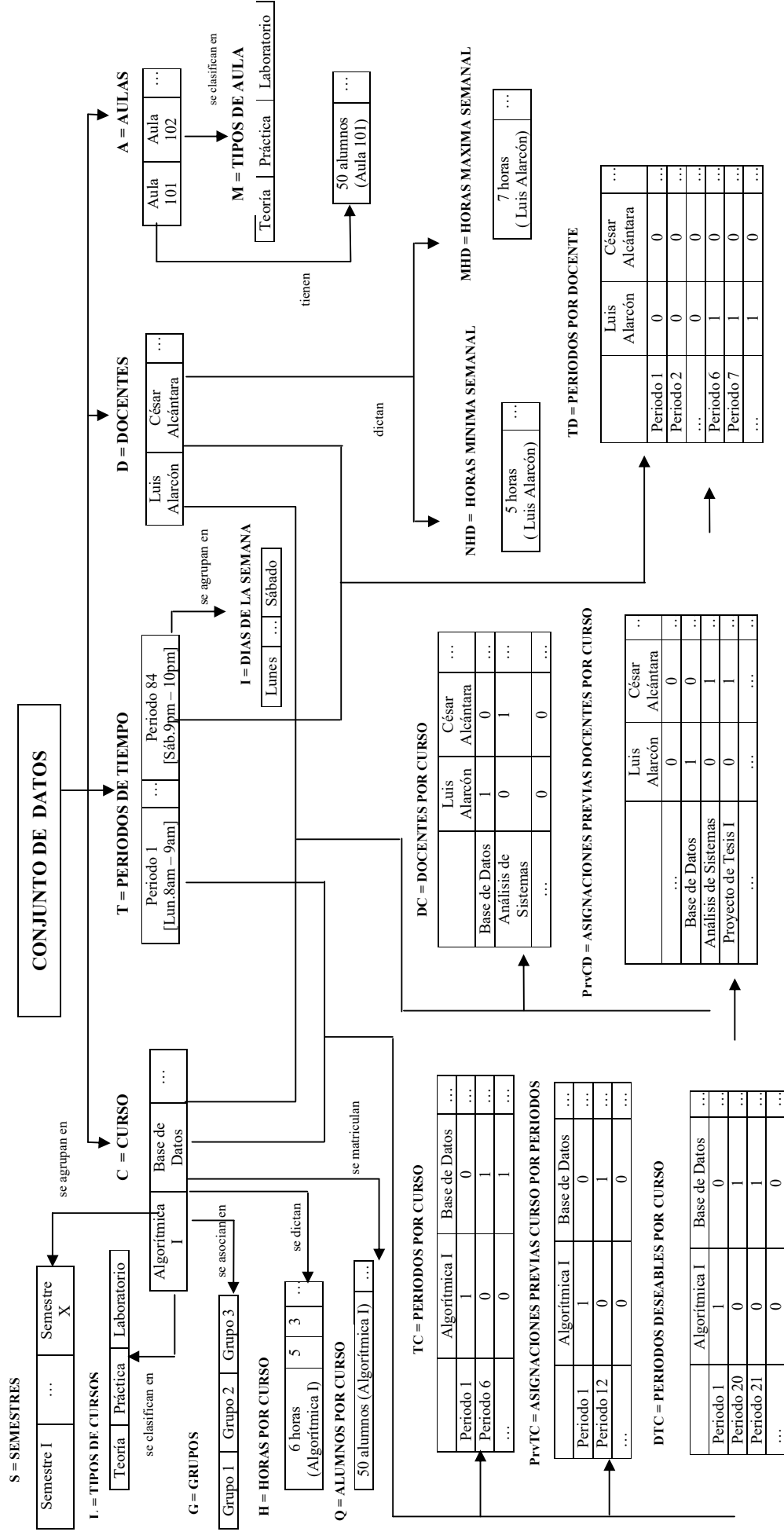
atrás exactamente igual que en un recorrido en profundidad, eliminando sobre la marcha los elementos que se hubieran añadido en cada fase. Cuando vuelve a un nodo que tiene uno o más vecinos sin explorar, prosigue el recorrido de una solución. [40]

- Algoritmo greedy o algoritmo voraz.- Un algoritmo voraz (también conocido como ávido o devorador) es aquel que, para resolver un determinado problema, sigue una meta heurística consistente en elegir la opción óptima en cada paso local con la esperanza de llegar a una solución general óptima. Este esquema algorítmico es el que menos dificultades plantea a la hora de diseñar y comprobar su funcionamiento. Normalmente se aplica a los problemas de optimización. [41]

Apéndice 2: Definición de acrónimos y abreviaturas

- AG.- Algoritmo Genético
- FISI.- Facultad de ingeniería de Sistemas e Informática
- AE.- Algoritmo evolutivo
- PLE.- Programación lineal entera
- UNMSM.- Universidad Nacional Mayor de San Marcos
- ADN.- Acido desoxirribonucleico
- GRASP.- Greedy Randomized Adaptive Search
- API.- Interfaz de programación de aplicaciones
- SUM.- Sistema Único de Matricula

Apéndice 3: Estructura de Datos del Modelo General



Apéndice 4: Descripción de paquetes del API JGAP

Para el desarrollo de la solución se utiliza la versión 3.4.4, siendo hasta este momento la última versión de JGAP. A continuación mostramos el conjunto de paquetes que lo conforman:

Cada uno de estos paquetes contiene interfaces, clases y excepciones que están referidos a la programación de algoritmos genéticos, por ello detallaremos una descripción y el contenido de cada uno de dichos paquetes. [42]

A continuación mostramos el conjunto de paquetes que lo conforman:

org.jgap: <i>Contiene las interfaces y clases que componen el núcleo de JGAP.</i>	
Interfaces	<i>FitnessEvaluator, Gene, GeneticOperator, IApplicationData, IBreeder, IBusinessKey, IChromosome, IChromosomePool, ICloneHandler, ICompareToHandler, ICompositeGene, IGeneConstraintChecker, IGeneticOperatorConstraint, IHandler, IInitializer, IJGAPFactory, INaturalSelector, IPersistentRepresentation, IUniversalRateCalculator, RandomGenerator</i>
Clases	<i>BaseChromosome, BaseGene, BaseGeneticOperator, BaseRateCalculator, BreederBase, BulkFitnessFunction, CachedFitnessFunction, Chromosome, Configuration, DefaultFitness Evaluator, DeltaFitnessEvaluator, FitnessFunction, Genotype, NaturalSelector, NaturalSelectorExt, Population</i>
Excepciones	<i>InvalidConfigurationException, UnsupportedRepresentationException</i>
org.jgap.audit: <i>Contiene funciones relacionadas con la evaluación del comportamiento de JGAP.</i>	
Interfaces	<i>IEvolutionMonitor</i>
Clases	<i>ChainedMonitors, Evaluator, FitnessImprovementMonitor, KeyedValue, KeyedValues, Keyed Values2D, PermutingConfiguration, TimedMonitor</i>
org.jgap.data: <i>Relacionado a la persistencia del estado de los objetos JGAP (genotipo, cromosomas)</i>	
Interfaces	<i>IDataCreators, IDataElement, IDataElementList</i>
Clases	<i>DataElement, DataElementList, DataElementsDocument, DataTreeBuilder, DocumentBuilderBase</i>
org.jgap.data.config: <i>Relacionado a la configuración de JGAP.</i>	
Interfaces	<i>Configurable, ConfigurationHandler, IConfigInfo</i>
Clases	<i>ConfigData, ConfigFileReader, ConfigProperty, ConfigWriter, MetaConfig,</i>

	<i>RootConfigurationHandler</i>
Excepciones	<i>ConfigException, MetaConfigException</i>
org.jgap.distr: Relacionado a la Computación distribuida de poblaciones de un Algoritmo Genético.	
Interfaces	<i>IPopulationMerger, IWorker</i>
Clases	<i>Breeder, Culture, CultureMemoryCell, DistributedGenotype, Master, MasterInfo, MasterListener, Problem, RequestDispatcher, Worker, WorkerCommand, WorkerListener</i>
org.jgap.distr.grid: Funcionalidad Grid para la construcción de una red de computación distribuida con el servidor, trabajador (s) y cliente (s).	
Interfaces	<i>IClientEvolveStrategy, IClientFeedback, IEvolveStrategy, IGenotypeInitializer, IGridClientMediator, IGridConfiguration, IGridMasterConnectionListener, IRequestSplitStrategy, IResultMergeStrategy, IWorkerEvolveStrategy, IWorkerReturnStrategy</i>
Clases	<i>DefaultClientFeedback, DefaultEvolveStrategy, DefaultGenotypeInitializer, DefaultWorkerReturnStrategy, DummyGridClientMediator, GridConfigurationBase, JGAPAdmin, JGAPClient, JGAPClientHandlerThread, JGAPRequest, JGAPResult, JGAPServer, JGAPWorker, JGAPWorkers, MessageContext, RequestSplitStrategy, SampleSplitStrategy</i>
org.jgap.distr.grid.common: Clases comunes para computación Grid.	
Clases	<i>BasicContext, ClientStatus, CommonConfiguration, ResultData, ResultVerification</i>
Excepciones	<i>NoWorkResultsFoundException, WorkRequestsSendException, WorkResultNotFoundException</i>
org.jgap.distr.grid.gp: Computación Grid para Programación Genética.	
Interfaces	<i>IClientEvolveStrategyGP, IClientFeedbackGP, IGenotypeInitializerGP, IGridConfigurationGP, IRequestSplitStrategyGP, IWorkerEvolveStrategyGP, IWorkerReturnStrategyGP</i>
Clases	<i>DefaultEvolveStrategyGP, GridConfigurationGPBase, JGAPClientGP, JGAPGPXStream, JGAPRequestGP, JGAPResultGP, JGAPWorkerGP, JGAPWorkersGP, NullClientFeedbackGP</i>
org.jgap.distr.grid.request: Mecanismos de solicitudes en Computación Grid.	
Clases	<i>Status, VersionInfo</i>

org.jgap.distr.grid.util: Utilitarios para Computación Grid.	
Clases	GridKit
org.jgap.distr.grid.wan: Acceso vía WAN.	
Clases	ReceivedPacket, RequestInformation, ResultInformation, WANData, WANUtils
org.jgap.eval: Auditoría del desempeño de JGAP con diferentes configuraciones.	
Clases	PopulationHistory
org.jgap.event: Contiene clases e interfaces relacionados con la generación y manejo de eventos JGAP.	
Interfaces	GeneticEventListener, IEventManager
Clases	EventManager, GeneticEvent
org.jgap.gp: Funcionalidad de programación genética (clases abstractas e interfaces).	
Interfaces	IComplexCommand, IGPCChromosome, IGPFitnessEvaluator, IGPGeneticOperator, IGPIInitStrategy, IGPPProgram, IMutateable, INaturalGPSelector, INodeValidator, IPopulationCreator, IProgramCreator, IUniqueCommand
Clases	BaseGPChromosome, CommandDynamicArity, CommandGene, CrossMethod, GPFitnessFunction, GPPopulationConverter, GPPProblem, GPPProgramBase, MathCommand
Excepciones	CommandGene.COMMAND_TYPE, IComplexCommand.Complexity
org.jgap.gp.function: Funciones para ser usados con Programación Genética.	
Interfaces	Abs.Compatible, Add.Compatible, Add3.Compatible, Add4.Compatible, ArcCosine.Compatible, ArcSine.Compatible, ArcTangent.Compatible, Ceil.Compatible, Cosine.Compatible, Divide.Compatible, Exp.Compatible, Floor.Compatible, Increment.Compatible, Log.Compatible, Modulo.Compatible, Multiply.Compatible, Multiply3.Compatible, Pow.Compatible, Sine.Compatible, Subtract.Compatible, Tangent.Compatible
Clases	Abs, Add, Add3, Add4, AddAndStore, AddAndStoreTerminal, ADF, And, ArcCosine, ArcSine, ArcTangent, Ceil, CharacterProvider, Cosine, CountMatrix, Divide, Equals, ExchangeMemory, Exp, Floor, ForLoop, ForXLoop, GreaterThan, If, IfDyn, IfElse, Increment, IncrementMemory, LesserThan, Log, Loop, Max, Min, Modulo, Multiply, Multiply3, Not, Or, Pop, Pow, Push, RandomGe

	<i>nerator, ReadFromMatrix, ReadTerminal, ReadTerminalIndexed, ReplaceInMatrix, ResetMatrix, Round, Sine, StoreTerminal, StoreTerminalIndexed, SubProgram, Subtract, Tangent, TransferMemory, Tupel, WriteToMatrix, Xor</i>
Excepciones	<i>CountMatrix.CountMode, CountMatrix.CountType, ReplaceInMatrix.ReplacementMode</i>
org.jgap.jp.impl: Clases de programación genética, no abstracta, sin interfaces.	
Clases	<i>BranchTypingCross, CommandFactory, DefaultGPFitnessEvaluator, DefaultPopulationCreator, DefaultProgramCreator, DeltaGPFitnessEvaluator, FitnessProportionateSelection, GPConfiguration, GPGenotype, GPPopulation, GPPProgram, GPPProgramFitnessComparator, GPPProgramInfo, JGAPTreeBranchRenderer, JGAPTreeNode, JGAPTreeNodeRenderer, ProgramChromosome, StaticPopulationCreator, TournamentSelector</i>
org.jgap.jp.terminal: Terminales para usar con programación genética.	
Clases	<i>Argument, Constant, False, NOP, Terminal, True, Variable</i>
org.jgap.gui: Interfaz gráfica de usuario para JGAP.	
Clases	<i>ConfigFrame, GUIManager</i>
org.jgap.impl: Proporciona implementaciones de diversos sub-componentes que pueden ser conectados ó customizados.	
Clases	<i>AveragingCrossoverOperator, BestChromosomesSelector, BooleanGene, BulkFitnessOffsetRemover, CauchyRandomGenerator, ChainOfSelectors, ChromosomePool, CompositeGene, CrossoverOperator, DefaultCloneHandler, DefaultCompareToHandler, DefaultConfiguration, DefaultCrossoverRateCalculator, DefaultInitializer, DefaultMutationRateCalculator, DoubleGene, FittestPopulationMerger, FixedBinaryGene, GABreeder, GaussianMutationOperator, GaussianRandomGenerator, GreedyCrossover, IntegerGene, InversionOperator, JGAPFactory, MapGene, MutationOperator, NumberGene, Pool, RangedSwappingMutationOperator, SetGene, StandardPostSelector, StockRandomGenerator, StringGene, SwappingMutationOperator, ThresholdSelector, TournamentSelector, TwoWayMutationOperator, WeightedRouletteSelector</i>
org.jgap.impl.fitness: Funcionalidad relacionada con el cálculo de valor de fitness.	
Clases	<i>TruthTableFitnessFunction</i>

org.jgap.impl.job : División de una solicitud de evolución en varios trabajo a fin de calcularlos de manera distribuida.	
Interfaces	<i>IEvolveJob , IJob , IPopulationSplitter</i>
Clases	<i>EvolveData , EvolveJob , EvolveResult , JobBase , JobData , JobResult , MaxFunction , SimpleJobConsumer , SimplePopulationSplitter</i>
org.jgap.impl.salesman : Manejo de problemas relacionados al problema del Viajero (TSP).	
Clases	<i>Salesman , SalesmanFitnessFunction</i>
org.jgap.supergenes : Supergenes que controlan aspectos estrechamente relacionados con el fenotipo.	
Interfaces	<i>Supergene , SupergeneValidator</i>
Clases	<i>AbstractSupergene , Validator</i>
org.jgap.util : Clases utilitarias comunes usados con JGAP.	
Interfaces	<i>ICloneable , ICommand</i>
Clases	<i>ChromosomeFitnessComparator, ClassKit, CommandResult, DateKit, FileKit, JarClassLoader, JarResources, LRUCache, MultiClassLoader, NetworkKit, NumberKit, PersistableObject, PluginDiscoverer, randomHotBits, randomJava, randomLCG, randomLEcuyer, randomMCG, randomX, StringKit, SystemKit, UniqueRandomGenerator, XStreamPassThruConverter</i>
Excepciones	<i>CloneException</i>
org.jgap.xml : Escritura / Lectura de los estados de objetos JGAP (genotipo, cromosomas, etc) a / de documentos XML.	
Clases	<i>XMLDocumentBuilder , XMLManager</i>
Excepciones	<i>GeneCreationException , ImproperXMLException</i>