

ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

DESARROLLO DE UN MÉTODO PARA LA RESOLUCIÓN DE
PROBLEMAS DE CALENDARIZACIÓN UTILIZANDO EL ENFOQUE
“OPTIMIZACIÓN DE COLONIA DE HORMIGAS”

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN
SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN

SILVA ARTIEDA PAOLA LORENA

pollipayi@yahoo.com

VALENCIA MOYA VICTOR OMAR

omarcoran@gmail.com

DIRECTOR: Dr. HUGO BANDA

hbanda@ieee.org

Quito, Agosto 2010

DECLARACIÓN

Nosotros Silva Artieda Paola Lorena y Valencia Moya Víctor Omar, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedemos nuestros derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

Silva Artieda Paola Lorena

Valencia Moya Víctor Omar

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Silva Artieda Paola Lorena y Valencia Moya Víctor Omar, bajo mi supervisión.

Dr. Hugo Banda

DIRECTOR DE PROYECTO

AGRADECIMIENTO

Agradezco en primer lugar a mis padres quienes me han sabido guiar en cada paso de mi vida, me han forjado en la mujer que soy gracias a sus sabios consejos y principios. Siempre estuvieron a mi lado durante toda mi carrera apoyándome en los buenos y malos momentos.

Al hombre quién en su momento me enseñó que la vida es de decisiones a ti FEHY.

A mi tía y mi prima por estar pendientes de mí y ayudarme a salir adelante.

A mis amigos, con quienes he compartido momentos agradables.

Al Doctor Hugo Banda, quien ha sabido guiarme durante todo el proyecto de titulación.

PAOLA SILVA

AGRADECIMIENTO

Un profundo agradecimiento a todos quienes colaboraron conmigo
en este proyecto

VICTOR VALENCIA

DEDICATORIA

Dedicó el presente trabajo a mis padres, hermanos, Fabi y a mi tía
Mirtha.

PAOLA.

DEDICATORIA

*Dedicado a mis padres, a mi amada esposa, y mi hermosa
niña*

VICTOR.

TABLA DE CONTENIDOS

1. CAPÍTULO 1. INTRODUCCIÓN	15
1.1. PROBLEMAS COMBINATORIOS	15
1.1.1. Definición de Problema de Optimización Combinatoria.....	15
1.1.2. Definición de Instancia de Problema Combinatorio	15
1.2. CALENDARIZACIÓN	17
1.2.1. Objetivo de la Calendarización	17
1.3. DESCRIPCIÓN DE ALGORITMOS PARA OPTIMIZACIÓN DE TAREAS DE CALENDARIZACIÓN.....	18
1.3.1. BÚSQUEDA TABÚ	18
1.3.2. ALGORITMOS GENÉTICOS.....	19
1.3.2.1. Ventajas de los Algoritmos Genéticos.....	20
1.3.2.2. Desventajas de los Algoritmos Genéticos.....	21
1.3.3. BÚSQUEDA LOCAL ITERATIVA (ILS).....	21
1.3.3.1. VISTA GRÁFICA.....	22
1.3.4. RECODIDO SIMULADO.....	23
1.3.4.1. Ventajas del Recocido Simulado.....	24
1.3.4.2. Desventajas del Recocido Simulado.....	25
1.3.5. PROGRAMACIÓN CON RESTRICCIONES.....	25
1.3.5.1. Problemas de Satisfacción de Restricciones (PSR).....	25
1.3.5.1.1. Propagación de Restricciones	26
1.3.5.1.2. Algoritmos de Búsqueda.....	27
1.4. MÉTODO DE OPTIMIZACIÓN “COLONIA DE HORMIGAS”.....	31
1.4.1. ACO INCLUYE DOS MECANISMOS	32
1.4.1.1. La evaporación de sendero.....	¡Error! Marcador no definido.
1.4.1.2. Acciones Demonio	32
1.4.2. ALGORTIMOS BASADOS EN ACO.....	33
1.4.2.1. ANT SYSTEM (AS).....	33
1.4.2.2. ANT COLONY SYSTEM (ACS)	33
1.4.2.3. MIN-MAX ANT SYSTEM (MMAS).....	35
1.4.2.4. APPROXIMATE NONDETERMINISTIC TREE SEARCH (ANTS)	37
1.4.2.4.1. Características de ANTS	37
2. CAPÍTULO 2. DESARROLLO DEL MÉTODO	38
2.1. CARACTERIZACIÓN DEL PROBLEMA	38
2.1.1. Modelamiento Basado en el Currículo:.....	39
2.1.2. Modelamiento Basado en los Datos de Inscripción:	39
2.2. DEFINICIÓN DE LOS TIPOS DE RESTRICCIONES	39
2.2.1. Restricciones Unitarias	42
2.2.2. Restricciones Binarias	42
2.2.3. Restricciones de Capacidad	42
2.2.4. Restricciones de repartición de eventos.....	42

2.2.5.	Restricciones de Agente.....	43
2.2.6.	Restricciones Duras.....	43
2.2.7.	Restricciones Suaves	43
2.3.	MODELAMIENTO MATEMÁTICO DEL PROBLEMA	46
2.3.1.	CONJUNTO DE DATOS:	46
2.3.2.	MATRICES DE ENTRADA:	49
2.3.3.	MATRICES DE SALIDA:	51
2.3.4.	RESTRICCIONES DURAS:.....	51
2.3.5.	RESTRICCIONES SUAVES.....	55
2.4.	CRITERIOS PARA ELEGIR EL ALGORITMO MÁS ADECUADO AL PROBLEMA.....	56
2.5.	HERRAMIENTAS DE IMPLEMENTACIÓN	60
3.	CAPÍTULO 3. CASO DE ESTUDIO.....	62
3.1.	CARACTERIZACIÓN DEL CASO DE ESTUDIO	62
3.2.	DEFINICION DE LOS TIPOS DE RESTRICCIONES EN EL CASO DE ESTUDIO.....	66
3.2.1.	Restricciones Duras.....	66
3.2.2.	Restricciones Suaves	67
3.2.3.	Restricciones de la interfaz de usuario	68
3.3.	MODELO MATEMATICO DEL CASO DE ESTUDIO.....	69
3.4.	ELECCION DEL ALGORITMO MÁS ADECUADO PARA EL CASO DE ESTUDIO.....	72
3.4.1.	Configuracion de los algoritmos.....	73
3.5.	IMPLEMENTACION DEL ALGORITMO	74
3.5.1.	Pseudo código del Algoritmo	75
3.5.2.	PRUEBAS	78
3.5.2.1.	Caso de Prueba: Definición de Datos	78
3.5.2.2.	Caso de Prueba: Datos Semestres.....	78
3.5.2.3.	Caso de Prueba: Datos Materias	79
3.5.2.4.	Caso de Prueba: Datos Profesores.....	79
3.5.2.5.	Caso de Prueba: Datos Aulas	80
3.5.2.6.	Caso de Prueba: Archivo	80
3.5.2.7.	Caso de Prueba: Búsqueda Profesor.....	81
3.5.2.8.	Caso de Prueba: Resultado Horario	81
4.	CAPÍTULO 4. CONCLUSIONES Y RECOMENDACIONES.....	83
4.1.	CONCLUSIONES	83
4.2.	RECOMENDACIONES	85
	BIBLIOGRAFÍA	87

ÍNDICE DE FÍGURAS

Figura 1.1 Ciclo Básico del Algoritmo Genético	20
Figura 1.2 Procedimiento General ILS	23
Figura 1.3 Hacer (x, y) y (y, x) arcos consistentes	27
Figura 1.4 Árbol de Búsqueda para las 4-reinas usando backtracking	29
Figura 1.5 Búsqueda de Árbol para 4-reinas usando Forward checking.....	30
Figura 2.1 Matriz Binaria Semestre-Materia.....	49
Figura 2.2 Nivel de Preferencia del Profesor para dictar una materia.....	50
Figura 2.3 Diagrama de Componentes para el Prototipo	61

ÍNDICE DE TABLAS

Tabla 1.1 Parámetros usados por los algoritmos	36
Tabla 3.1 Configuración de los Algoritmos	73
Tabla 3.2 Caso de Prueba: Definición de Datos.....	78
Tabla 3.3 Caso de Prueba: Datos Semestres	78
Tabla 3.4 Caso de Prueba: Datos Materias	79
Tabla 3.5 Caso de Prueba: Datos Profesores	79
Tabla 3.6 Caso de Prueba: Datos Aulas	80
Tabla 3.7 Caso de Prueba: Archivo.....	80
Tabla 3.8 Caso de Prueba: Búsqueda Profesor	81
Tabla 3.9 Caso de Prueba: Resultado Horario.....	81

RESUMEN

Este trabajo, enfrenta el problema de automatizar la generación de horarios en el ambiente universitario. Esta es una tarea que debido a su complejidad se ha realizado de manera manual en la mayoría de universidades a nivel mundial.

Esto empezó a cambiar a mediados de los años 90, a medida que se desarrollaban mejoras en las heurísticas existentes para enfrentar los problemas combinatorios, dentro de los cuales se enmarca la generación automática de horarios.

La necesidad de automatizar este aspecto de la vida universitaria surge debido al crecimiento constante tanto del número de alumnos, como de programas académicos que las universidades ofrecen. Estos aumentos han traído una serie de conflictos, al momento de crear los horarios, mas aun si como en el caso de la EPN existen altos índices de repetición entre los alumnos.

Es por esto que este trabajo se enfoca en realizar un modelo que siga de cerca la cultura universitaria de la EPN y de manera particular el modo de realizar horarios en la Facultad de Sistemas.

Creemos que de esta manera las maquinas se encargaran de la mayor parte del trabajo, liberando a las personas para que se hagan cargo de los problemas más difíciles y especiales.

PRESENTACIÓN

Este proyecto de titulación plantea la elaboración de un modelo matemático que refleje la mayoría de las prácticas al momento de elaborar horarios, en la FIS de la EPN, así como también la elaboración de una aplicación acompañante que haga uso del mismo.

Se espera que la aplicación ayude en la automatización de una de las tareas más complejas dentro de la administración universitaria.

Como producto final se tiene una aplicación que se ejecuta en ambientes WINDOWS, y que se ha desarrollado usando C# como lenguaje de programación.

A continuación se detalla el contenido de todos los capítulos:

CAPÍTULO 1. INTRODUCCIÓN

En este capítulo se realiza una breve introducción a los conceptos de problemas combinatorios y calendarización. También se realiza una breve introducción a los algoritmos y heurísticas más usados para resolver este tipo de problemas, se presentan también las ventajas de usar cada uno de estos enfoques. Así mismo se resumen para el lector, los conceptos fundamentales de la satisfacción de restricciones y de las Metaheurísticas que conforma ANT COLONY.

CAPÍTULO 2. DESARROLLO DEL MÉTODO

En este capítulo se realiza la caracterización del problema en general, se describen con detalles cada uno de los tipos de restricciones que se encuentran dentro del problema y se realiza una aproximación a lo que sería un modelo en general de la calendarización universitaria basada en el currículo. Dentro del modelo se introducen los conceptos fundamentales de los diferentes tipos de datos que definen el problema, así como también cada una de las restricciones que serán aplicadas a estos.

CAPÍTULO 3. CASO DE ESTUDIO

En este capítulo se realiza la caracterización del caso de estudio, que es la manera de generación de horarios en la FIS de la EPN.

Se realiza una descripción de la cultura en la facultad, los recursos con los que cuenta y a partir de todos estos datos se genera un modelo matemático que se ajusta lo más posible a todas estas características.

Se realiza una descripción de los algoritmos de implementación, así como también la descripción de la arquitectura usada dentro de la aplicación acompañante.

CAPÍTULO 4. CONCLUSIONES Y RECOMENDACIONES

En este capítulo se extraen y presentan las conclusiones y recomendaciones que se han generado después de la elaboración de este proyecto.

1. CAPÍTULO 1. INTRODUCCIÓN

1.1. PROBLEMAS COMBINATORIOS

Un problema de optimización combinatoria o problema combinatorio es especificado por un conjunto de instancias del problema y es un problema de minimización o maximización.

Por **problema** se entiende, una pregunta general a ser contestada, teniendo muchas variables y parámetros con valores no especificados.

La **instancia** del problema a su vez se entiende como un caso particular del problema, con valores especificados para todos los parámetros y variables.

Para entender mejor estos planteamientos, pensemos en el conocido problema del vendedor viajero, en términos generales, este problema se trata del vendedor que debe visitar todas las ciudades que necesita solamente una vez y hacerlo en la ruta más corta, hasta volver a su punto de inicio, nótese que no se han explicitado los nombres de las ciudades, ni las distancias entre ellas, es decir, este planteamiento es válido para cualquier caso.

La instancia se puede entender mejor si pensamos en un caso particular, tomando nuevamente como ejemplo el vendedor viajero, una instancia de este problema será el caso de un vendedor que deba visitar las capitales de todas las provincias del Ecuador, partiendo desde Quito, hacerlo en la ruta más corta y volver a la capital una vez finalizado su recorrido.

Esta es una manera informal de visualizar estas cuestiones pero como saben debemos añadirles el lenguaje formal de las matemáticas.

1.1.1. Definición de Problema de Optimización Combinatoria

Un problema combinatorio se define como el conjunto P de todas las instancias que están asociadas a estas soluciones.

1.1.2. Definición de Instancia de Problema Combinatorio

La instancia de un problema dentro de un espacio combinatorio es un par (V, f) donde V es el conjunto de soluciones viables para la instancia y f es la función objetivo o de costo, la cual asigna a cada solución viable un valor dentro de los reales, es decir es una aplicación $f: V \Rightarrow \mathcal{R}$, la meta es hallar una solución

globalmente óptima, de tal manera que para un $s \in V$ se satisfaga que $f(s) \leq f(i)$, $\forall i \in V$.

En la definición se ha planteado un problema de minimización, en el caso de un problema de maximización se tiene un planteamiento similar excepto que la solución globalmente óptima debe cumplir con $f(s) \geq f(i)$, $\forall i \in V$.

Es debido remarcar que las instancias no son dadas explícitamente, es decir no son expresadas directamente mediante el par (V, f) es decir todas las soluciones viables y sus costos, sino que más bien se expresan en un conjunto de valores relacionados directamente con los parámetros que han sido usados para modelar el problema. En muchos casos además de encontrar una solución óptima debemos además hallar una solución que satisfaga una serie de restricciones dentro del espacio de solución, es decir una solución podría no ser válida aunque sea globalmente mínima (o máxima).

Este aspecto está tomado en cuenta dentro de la definición hecha en los párrafos anteriores, pero de manera implícita al decir que V **es el conjunto de las soluciones viables o factibles**, una de las maneras de acotar V es mediante el uso de restricciones.

Si deseamos que las restricciones sean tomadas en cuenta explícitamente dentro de la definición de la instancia de problema de optimización combinatoria podemos incluirlas de la siguiente manera:

La instancia de un problema combinatorio es una terna (S, f, θ) donde S es el conjunto de soluciones candidatas para la instancia, θ es un conjunto de restricciones y f es la función objetivo o de costo, la cual asigna a cada solución candidata un valor dentro de los reales, es decir es una aplicación $f: S \Rightarrow \mathcal{R}$. Las soluciones que pertenezcan a \check{S} de manera que $\check{S} \subseteq S$ y que cumplan con todas las restricciones dentro de θ son denominadas soluciones viables o factibles y \check{S} se denomina el conjunto de soluciones viables.

La meta es hallar una solución globalmente óptima, de tal manera que para un $s \in \check{S}$ se satisfaga que $f(s) \leq f(i)$, $\forall i \in \check{S}$. De similar manera si se trata con un problema de maximización la solución globalmente óptima debe cumplir con $f(s) \geq f(i)$, $\forall i \in \check{S}$.

1.2. CALENDARIZACIÓN

Muchas veces la programación de horarios o calendarización¹ de horarios, la secuenciación y la calendarización de actividades, se usan como sinónimos. Sin embargo puede haber algunas diferencias entre los términos.

Una calendarización de horarios, nos muestra cuando ciertos eventos van a ocurrir, y no implica directamente la asignación de recursos, así puede servirnos de ejemplo la calendarización de los horarios del transporte público, esta nos muestra cuando los viajes hacia una ruta o rutas van a efectuarse, y nada nos dice sobre los buses o choferes con los que se efectuará dichos viajes. La asignación de vehículos y choferes forma parte de la calendarización de actividades.

Una secuencia es simplemente el orden en que las tareas se llevan a cabo, como ejemplo se puede tomar el orden con que las tareas se procesan a través de la maquinaria de una empresa o fabrica, si los trabajos pasan por las máquinas siempre exactamente con el mismo orden entonces es una secuencia.

La calendarización de actividades, normalmente incluye toda la información temporal y espacial necesaria para lleva a cabo el proceso. Esto incluye los tiempos en los cuales las actividades tendrán lugar, declaraciones de cuáles recursos serán asignados y donde. Por esto se completa con planes de trabajos para personas y máquinas individualmente.

1.2.1. Objetivo de la Calendarización

En el sentido más amplio el objetivo de la calendarización es el de resolver problemas prácticos, relacionados con la asignación de recursos, en el espacio-tiempo, el cual está sujeto a una serie de restricciones. Haciendo uso o desarrollando todas las herramientas que sean necesarias.

Definiendo, todos estos conceptos más formalmente tenemos:

¹ Calendarización: Ver glosario

Calendarización de actividades:

Es la asignación de recursos a objetos sujeta a restricciones, siendo estos colocados en el espacio-tiempo de una manera que minimice el costo total de algunos recursos usados.

Calendarización de horarios:

Es la asignación, sujeta a restricciones, de recursos a objetos siendo colocados en el espacio- tiempo, de tal manera que se satisfagan tanto como sea posible, un conjunto de objetivos deseados.

Secuenciación de Actividades:

Es la construcción, sujeta a restricciones, de un orden particular en el cual las actividades serán llevadas a cabo o a su vez objetos serán dispuestos en ese orden particular en representación de una solución.

1.3. DESCRIPCIÓN DE ALGORITMOS PARA OPTIMIZACIÓN DE TAREAS DE CALENDARIZACIÓN

1.3.1. BÚSQUEDA TABÚ

La búsqueda local emplea la idea que una solución dada, S puede ser mejorada al hacer pequeños cambios. Estas soluciones obtenidas modificando la solución S son llamadas vecinos de S . El algoritmo de búsqueda local empieza con alguna solución inicial y se mueve de vecino a vecino tanto como sea posible, mientras decrezca el valor de la función objetivo. El principal problema con esta estrategia es escapar de un mínimo local donde la búsqueda no puede encontrar ninguna solución vecina que reduzca el valor de la función objetivo. Diferentes estrategias han sido propuestas para salvar este problema. Una de las estrategias más eficientes es la búsqueda tabú. La búsqueda Tabú permite la búsqueda, para explorar soluciones que no reducen el valor de la función objetivo solo en los casos donde estas soluciones no son prohibidas. Esto es usualmente obtenido manteniendo pistas de las últimas soluciones, en términos de las acciones usadas para transformar una solución en la siguiente. Cuando una acción es

llevada a cabo está es considerada tabú por la siguientes T iteraciones, donde T es la duración del estatus de tabú. Una solución está prohibida si esta se obtiene al aplicar una acción tabú a la solución en curso. La meta heurística ² Búsqueda Tabú ha sido definida por Fred Glover.

1.3.2. ALGORITMOS GENÉTICOS

Los algoritmos genéticos son una familia de modelos computacionales inspirados en la evolución. Estos algoritmos codifican una solución potencial a un problema específico en un cromosoma simple como la estructura de datos y aplicable a los operadores de recombinación de estas estructuras a fin de preservar la información crítica.

Los algoritmos genéticos son con frecuencia vistos como funciones de optimización, aunque el alcance del problema para el cual los algoritmos genéticos han sido aplicados es bastante amplio.

Expuesto concisamente, un algoritmo genético (o AG para abreviar) es una técnica de programación que imita a la evolución biológica como estrategia para resolver problemas. Dado un problema específico a resolver, la entrada del AG 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 candidata. Estas candidatas pueden ser soluciones que ya se sabe que funcionan, con el objetivo de que el AG las mejore, pero se suelen generar aleatoriamente.

Luego el AG evalúa cada candidata de acuerdo con la función de aptitud. En un acervo de candidatas generadas aleatoriamente, por supuesto, la mayoría no funcionarán en absoluto, y serán eliminadas. Sin embargo, por puro azar, unas pocas pueden ser prometedoras -pueden mostrar actividad, aunque sólo sea actividad débil e imperfecta, hacia la solución del problema.

Estas candidatas prometedoras se conservan y se les permite reproducirse. Se realizan múltiples copias de ellas, pero las copias no son perfectas; se introducen cambios aleatorios durante el proceso de copia. Luego, esta descendencia digital prosigue con la siguiente generación, formando un nuevo acervo de soluciones

² Meta heurística: ver glosario

candidatas, y son sometidas a una ronda de evaluación de aptitud. Las candidatas que han empeorado o no han mejorado con los cambios en su código son eliminadas de nuevo; pero, de nuevo, por puro azar, las variaciones aleatorias introducidas en la población pueden haber mejorado a algunos individuos, convirtiéndolos en mejores soluciones del problema, más completas o más eficientes. De nuevo, se seleccionan y copian estos individuos vencedores hacia la siguiente generación con cambios aleatorios, y el proceso se repite. Las expectativas son que la aptitud media de la población se incrementará en cada ronda y, por tanto, repitiendo este proceso cientos o miles de rondas, pueden descubrirse soluciones muy buenas del problema.

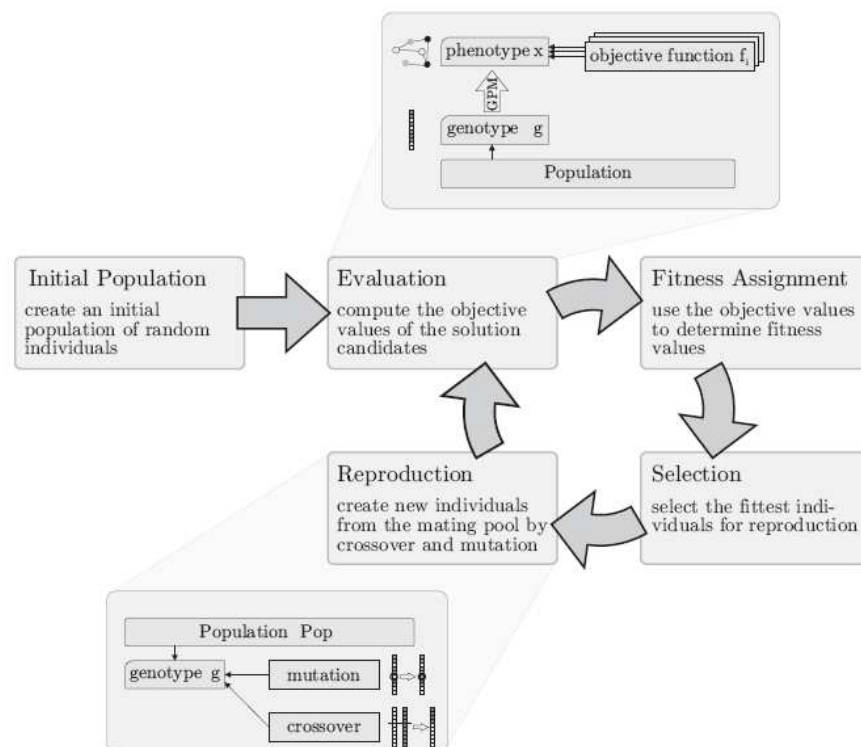


Figura 1.1 Ciclo Básico del Algoritmo Genético³

1.3.2.1. Ventajas de los Algoritmos Genéticos

El primer y más importante punto es que los algoritmos genéticos son intrínsecamente paralelos. La mayoría de los otros algoritmos son en serie y sólo pueden explorar el espacio de soluciones hacia una solución en una dirección al

³ Tomado del libro Global Optimization Algorithms de Tomás Weise Versión 2009-06-26

mismo tiempo, y si la solución que descubren resulta subóptima, no se puede hacer otra cosa que abandonar todo el trabajo hecho y empezar de nuevo. Sin embargo, ya que los AGs tienen descendencia múltiple, pueden explorar el espacio de soluciones en múltiples direcciones a la vez. Si un camino resulta ser un callejón sin salida, pueden eliminarlo fácilmente y continuar el trabajo en avenidas más prometedoras, dándoles una mayor probabilidad en cada ejecución de encontrar la solución.

1.3.2.2. Desventajas de los Algoritmos Genéticos

El problema de cómo escribir la función de aptitud debe considerarse cuidadosamente para que se pueda alcanzar una mayor aptitud y verdaderamente signifique una solución mejor para el problema dado. Si se elige mal una función de aptitud o se define de manera inexacta, puede que el algoritmo genético será incapaz de encontrar una solución al problema, o puede acabar resolviendo el problema equivocado.

1.3.3. BÚSQUEDA LOCAL ITERATIVA (ILS)

La esencia de la meta heurística búsqueda local iterativa se puede dar en una cáscara de nuez: uno construye iterativamente una secuencia de soluciones generadas por la heurística⁴ incrustada, lo que conduce a mejores soluciones que si se hubieran usado intentos al azar de esa heurística. Esta idea simple tiene una larga historia, y su redescubrimiento por muchos autores le ha designado muchos nombres diferentes como: Descenso Iterativo, Cadenas de Markov paso-amplio, Iterativa de Lin-Kernighan, Optimización Local Encadenada.

Hay dos puntos principales que hacen de un algoritmo un ILS:

- Debe existir una cadena simple que está siendo seguida (esto entonces excluye los algoritmos basados en población).
- La búsqueda para mejores soluciones ocurre en un espacio reducido definido por la salida de una heurística de caja negra.

⁴ Heurística: ver glosario

En la práctica, búsqueda local ha sido la heurística más frecuentemente usada de las heurísticas incrustadas, pero a decir verdad cualquier optimizador puede ser usado, sea determinística o no.

ILS explora la búsqueda de mínimos locales (denominada como S^*) para alguna heurística incrustada dada, llamada búsqueda local en lo subsiguiente. ILS logra esto heurísticamente de la siguiente manera:

Dado un s^* actual, primero aplicamos un cambio o perturbación que conduce a un estado intermedio s' (que pertenece a S). Entonces la búsqueda local es aplicada a s' y se alcanza una solución s^{**} en S^* . Si s^{**} pasa una prueba de aceptación, esto se convierte en el próximo elemento en el camino de S^* ; de otro modo se retorna a s^* . El camino resultante es un caso de búsqueda estocástica en S^* , pero donde las vecindades nunca son explícitamente introducidas. Este procedimiento ILS debería conducir a un muestreo bien sesgado, en tanto las perturbaciones no sean ni muy pequeñas ni muy grandes.

Si son muy pequeñas, con frecuencia se regresara a s^* y pocas nuevas soluciones serán exploradas. Si por el contrario las perturbaciones son muy grandes, s' será aleatorio, no habrá sesgamiento en el muestreo y se recuperará un tipo de algoritmo de reinicio al azar.

1.3.3.1. VISTA GRÁFICA

El procedimiento general de ILS es ilustrado gráficamente en la siguiente figura 1.2. Se debe notar que generalmente el camino ILS no es reversible; en particular uno puede algunas veces ser capaz de pasar de $s(1)^*$ a $s(2)^*$ pero no $s(2)^*$ a $s(1)^*$. Sin embargo esta característica desafortunada no impide que ILS sea un procedimiento efectivo en la práctica.

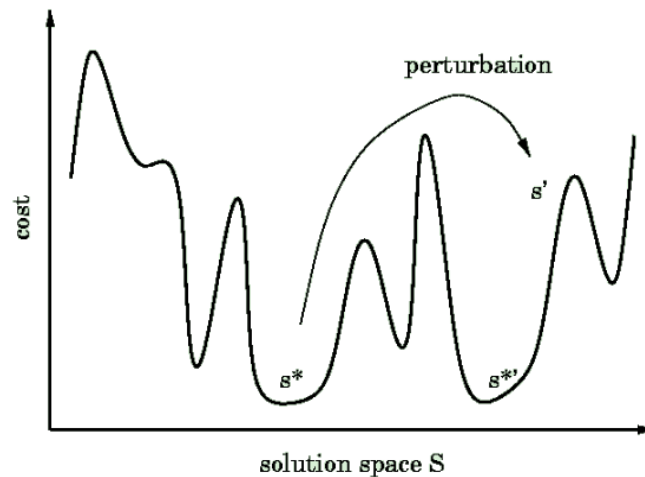


Figura 1.2 Procedimiento General ILS⁵

Dado que las perturbaciones determinísticas pueden conducir a ciclos cortos (por ejemplo de longitud 2), se debería asignar al azar las perturbaciones o hacerlas adaptivas, para de esta manera evitar este tipo de recurrencia.

1.3.4. RECOCIDO SIMULADO

Se dice que el recocido simulado es la más vieja entre las meta heurísticas y seguramente uno de los primeros algoritmos que tienen una estrategia explícita para evitar mínimos locales. Los orígenes del algoritmo se encuentran en mecánica estadística (Algoritmos Metrópolis⁶) y este fue el primer algoritmo de búsqueda presentado para problemas CO (Combinatorial Optimization). La idea fundamental es permitir movimientos que resulten en soluciones de peor calidad que la solución actual (movimientos colina arriba), con el fin de escapar de mínimos locales. La probabilidad de hacer tal movimiento es disminuído durante la búsqueda.

El algoritmo empieza generando una solución inicial (generada al azar o construida heurísticamente) e inicializando el supuesto parámetro de temperatura denominado T.

⁵ Tomado del libro Iterated Local Search & Variable Neighborhood Search de T. Stuetzle, 2003

⁶ Algoritmos Metrópolis es un método Markov chain Monte Carlo (son una clase de algoritmos para muestreo de distribuciones de probabilidad) para obtener una secuencia de muestras aleatorias de una distribución de probabilidad en el cual el muestreo directo es difícil. Esta secuencia puede ser utilizada para aproximar la distribución (ej. Para generar un histograma), o para calcular una integral (como un valor esperado).

Entonces, el siguiente se repite hasta que la condición de terminación se satisfaga: Una solución S' de la vecindad $N(s)$ de la solución S es una muestra aleatoria y está es aceptada como una nueva solución actual en función de $f(s)$, $f(s')$ y T . S' reemplaza a S si $f(s') < f(s)$ o, en caso $f(s') \geq f(s)$, con una probabilidad la cual es una función de T y $f(s') - f(s)$. La probabilidad es generalmente calculada siguiendo la distribución Boltzmann $\exp(-f(s')-f(s))/T$.

La temperatura T es reducida durante el proceso de búsqueda, por lo tanto al inicio de la búsqueda la probabilidad de aceptar movimientos hacia arriba es alta y gradualmente decrece, convergiendo a un algoritmo de mejoramiento iterativo simple. Este proceso es análogo al proceso de recocido de metales y vidrios, los cuales asumen una configuración baja de energía cuando enfrían con un apropiado programa de enfriamiento. Respecto al proceso de búsqueda, esto significa que el algoritmo es el resultado de dos estrategias combinadas: la caminata aleatoria y la mejora iterativa.

En la primera fase de la búsqueda, el prejuicio hacia las mejoras es bajo y esto permite la exploración del espacio de búsqueda; este componente irregular es lentamente reducido por lo tanto direccionando la búsqueda para converger a un mínimo local. La probabilidad de aceptación de movimientos hacia arriba es controlado por dos factores: la diferencia de la función objetivo y la temperatura. Por una parte, temperatura fija, la diferencia más alta $f(s') - f(s)$, la probabilidad más baja para aceptar un movimiento de S a S' . Por otra parte, cuanto más alto el valor de T , más alta la probabilidad de movimientos hacia arriba.

1.3.4.1. Ventajas del Recocido Simulado

El recocido simulado (da algunos supuestos cuenta sobre el enfriamiento de la temperatura) se ha demostrado que converge a la solución óptima de un problema.

Una implementación sencilla del algoritmo hace que sea muy fácil de adaptar un método de búsqueda local (por ejemplo, un mejor progreso en la búsqueda local) a un algoritmo de recocido simulado, usualmente aplicando la búsqueda local con resultados mucho mejores.

1.3.4.2. Desventajas del Recocido Simulado

Aunque se ha demostrado que el algoritmo converge al óptimo, este converge en un tiempo infinito. No solo por esta razón, si no debido a que se debe enfriar lentamente, el algoritmo no es usualmente más rápido que sus contemporáneos.

1.3.5. PROGRAMACIÓN CON RESTRICCIONES

La Programación con Restricciones es un paradigma de programación en el que las relaciones entre las variables se expresan en forma de restricciones. Las restricciones difieren de las primitivas comunes de lenguajes de programación ya que no se especifica un paso o secuencia de tareas a ejecutar, sino más bien las propiedades de una solución que se encontró. Esto hace de la programación con restricciones una forma de programación declarativa.

Las restricciones usadas en programación con restricciones son de varios tipos: los que se utilizan en Problemas de Satisfacción de Restricciones (e.g. "A o B es verdadero"), los resueltos por el algoritmo simple (e.g. " $X < 5$ "). Las restricciones son por lo general integradas dentro de un lenguaje de programación.

1.3.5.1. Problemas de Satisfacción de Restricciones (PSR)

Los problemas de Satisfacción de Restricciones o CSP (Constraint Satisfaction Problems) son problemas matemáticos definidos como un conjunto de objetos cuyo estado debe satisfacer un número de restricciones o limitaciones. Los CSP representan las entidades en un problema como una colección homogénea de restricciones finitas sobre las variables, la cual es resuelta por métodos de satisfacción de restricciones.

Los CSP son objeto de intensa investigación en inteligencia artificial e investigación de operaciones, ya que la regularidad en su formulación provee una base común para analizar y resolver problemas de muchas familias no relacionadas. CSP con frecuencia presentan alta complejidad, requiriendo una combinación de heurísticas y métodos de búsqueda combinatoria para resolverlos en un tiempo razonable.

Los CSP sobre dominios finitos son típicamente resueltos usando una forma de búsqueda. Las técnicas más usadas son: *Propagación de Restricciones*, *Variantes de Backtracking* y *Búsqueda Local*.

1.3.5.1.1. Propagación de Restricciones

Las técnicas de *Propagación de Restricciones* son métodos usados para modificar un problema de satisfacción de restricciones. Más precisamente, son métodos que cumplen una forma de consistencia local, las cuales son condiciones relacionadas a la consistencia de un grupo de variables y/o restricciones.

En la satisfacción de restricciones, las condiciones de consistencia local son propiedades de los problemas de satisfacción de restricciones relacionadas a la consistencia de subconjuntos de variables o restricciones. Varias condiciones existen, las más conocidas son: consistencia de nodo, consistencia de arco y consistencia de camino.

Consistencia de Arco

Brevemente, un Problema de Satisfacción de Restricciones (CSP) consiste de:

- Un conjunto de variables $X = \{x_1, \dots, x_n\}$
- Para cada variable x_i , un conjunto finito de posibles valores (su dominio D)

Si hay una restricción binaria C_{ij} entre las variables x_i y x_j entonces el $arc^2(x_i, x_j)$ es consistencia de arco si para cada valor $a \in D_i$, hay un valor $b \in D_j$ tal que la asignación $x_i = a$ y $x_j = b$ satisfacen la restricción C_{ij} .

Cualquier valor $a \in D_i$ para el cual este no es verdadero, no existe tal valor de b , puede seguramente ser removido desde D_i , esto hace que no pueda ser parte de ninguna solución de restricción: remover todos los valores como a hacen del arco (x_i, x_j) un arco consistente. Se debe solo verificar los valores de x_i ; todavía pueden haber valores en el dominio de x_j los cuales pueden ser removidos si invertimos la operación y hacemos al arco (x_j, x_i) un arco consistente. La Figura 1.3 (a) el dominio original de x y y . En (b), (x, y) ha sido hecho arco consistente. En (c), ambos (x, y) y (y, x) han sido hechos arcos consistentes. (Hay que tener en cuenta que si una restricción binaria es representada por una matriz, como se demostró antes, hace que las restricciones de arco consistente en ambas

direcciones efectivamente remuevan cualquier valor del dominio de las dos variables para lo cual las correspondientes fila o columna de la matriz son todas ceros.)

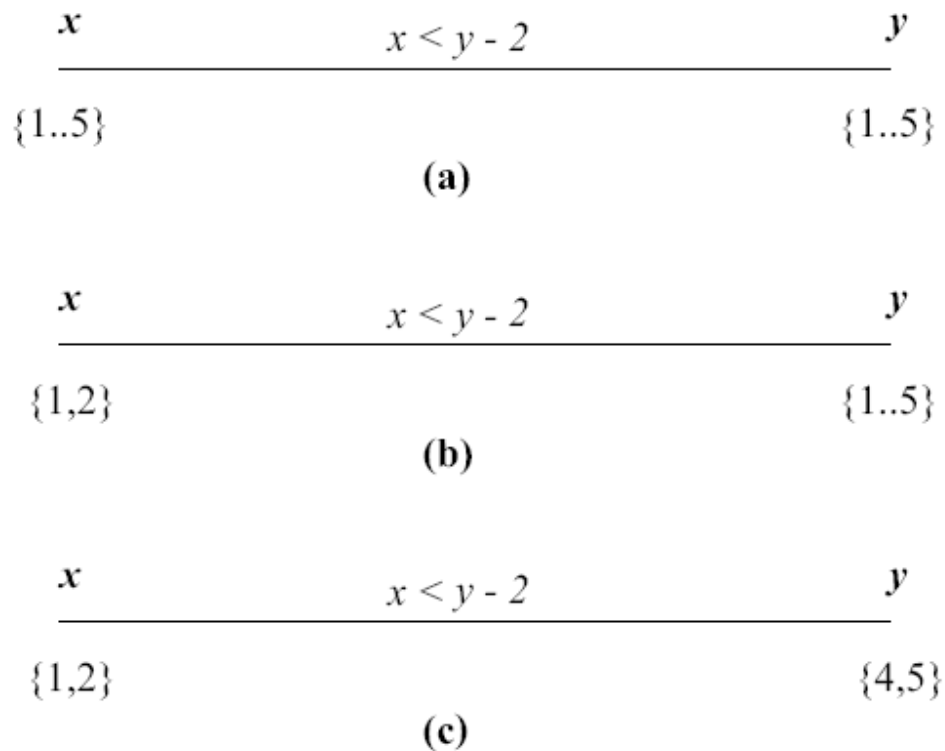


Figura 1.3 Hacer (x, y) y (y, x) arcos consistentes⁷

Si cada arco en un CSP binario es hecho arco consistente, entonces el problema completo se dice que es de arco consistente. El problema de consistencia de arcos es con frecuencia realizado como un escenario de pre-procesamiento: reduciendo los tamaños de algunos dominios debería hacer que el problema sea más fácil de resolver.

1.3.5.1.2. Algoritmos de Búsqueda

Vamos a considerar solo dos algoritmos de búsqueda sistemática: Un simple algoritmo de backtracking y el algoritmo Forward checking. Ambos algoritmos instancian cada variable en turno, y aumentan una solución parcial consistente de las variables ya consideradas, con sus valores asignados; estas son llamadas

⁷ Tomado del Tutorial Constraint Programming por Bárbara Smith, Abril 1995

variables del pasado. Las variables que aun no han sido instanciadas son las variables futuras.

En el **algoritmo *backtracking***, la variable actual es asignada a un valor desde su dominio. Esta asignación es entonces verificada otra vez por la solución parcial actual; si cualquiera de las restricciones entre estas variables y las variables pasadas es violada, la asignación es abandonada y otro valor para la variable actual es elegido. Si todos los valores para la variable actual han sido probados, el algoritmo retrocede a la variable previa y asigna este a un nuevo valor. Si una solución completa es encontrada, por ejemplo un valor a sido asignado a cada variable, el programa puede terminar, si solo una solución es requerida, o continuar para encontrar nuevas soluciones. Si no hay ninguna solución, el algoritmo finaliza cuando todas las posibilidades han sido consideradas.

Un ejemplo de construcción de árbol de búsqueda por el algoritmo *backtracking* es mostrado en la Figura. 1.4, usando el problema de las 4 reinas. Como un CSP, este problema tiene 4 variables, representando las filas del tablero de ajedrez, y cada variable tiene dominio $\{1, \dots, 4\}$ representando las 4 columnas. Sin embargo, es más fácil seguir el proceso de la búsqueda si la representación del tablero de ajedrez es usada: una Q sobre un cuadrado particular debe ser tomada en el sentido de que la variable correspondiente a esa fila ha sido asignada al valor correspondiente a esa columna. Callejones sin salida, donde el algoritmo ha retrocedido a una selección previa, están marcados por cruces, y la solución eventualmente encontrada está marcada por un visto.

El algoritmo *backtracking* solo verifica las restricciones entre la variable actual y las variables pasadas. El algoritmo *Forward checking*, por otra parte, verifica las restricciones entre las variables actuales (y pasadas) y las variables futuras. Cuando un valor es asignado a la variable actual, cualquier valor en el dominio de una variable futura que entra en conflicto con esta asignación es (temporalmente) removida del dominio.

reina de la primera fila deba ser removida. El árbol de búsqueda completo construido para forward checking para este problema es mostrado en la Figura 1.5. Cuadrados con cruces denotan valores removidos de los dominios de variables futuras por las asignaciones pasadas y actuales.

Forward checking trabaja más cuando cada asignación es añadida a la solución parcial actual, a fin de reducir el tamaño del árbol de búsqueda y así (con suerte) reducir la cantidad total de trabajo realizado. De hecho, en forward checking las 4-reinas transmiten la misma cantidad de trabajo como el algoritmo backtracking en la verificación de la consistencia; no hay suficiente alcance como tal, en una pequeña búsqueda de árbol para el podamiento temprano de ramales grandes.

Sin embargo, el siguiente ejemplo artificial muestra que el forward checking puede guardar una cantidad arbitraria de trabajo comparado con el simple backtracking: supongamos que tenemos variables $x_1, x_2, x_3, \dots, x_n$ donde x_1, x_2, x_n todas tienen dominio $\{1, 2\}$ y las restricciones sobre estas tres variables son que estas deben tener diferentes valores.

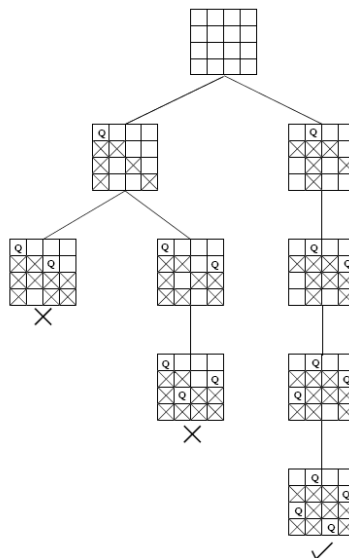


Figura 1.5 Búsqueda de Árbol para 4-reinas usando Forward checking⁹

Forward checking, por otro lado, descubrirá que no hay ningún valor que quede en el dominio de x_n tan pronto como los valores han sido asignados a x_1 y x_2 :

⁹ Tomado del Tutorial Constraint Programming por Bárbara Smith, Abril 1995

Esto nunca considerara la asignación de valores a las variables restantes. Forward checking no es la única manera de evitar este tipo de sandez, y esto puede tener dificultades en sí, en general, sin embargo, se lleva a cabo razonablemente en comparación con otros algoritmos, cuando combinamos con buenas heurísticas, y esto es casi siempre una elección mucho mejor que un simple backtracking.

1.4. MÉTODO DE OPTIMIZACIÓN “COLONIA DE HORMIGAS”

Optimización Colonia de Hormigas (ACO: Ant Colony Optimization) es un paradigma para diseñar algoritmos meta heurísticos¹⁰ para los problemas de optimización combinatorios. La meta heurística Optimización Colonia de Hormigas fue descrita primero por Marco Dorigo en su tesis de PHD, y fue inicialmente aplicada al problema del vendedor viajero (TSP). El problema del vendedor viajero es un problema de optimización combinatoria. Un vendedor gasta su tiempo visitando n ciudades (o nodos) en ciclos. En un viaje este visita cada ciudad una sola vez, y termina donde empezó. ¿En qué orden debe visitarlos para minimizar la distancia recorrida?

Esto ha sido desde entonces aplicado a muchos problemas de optimización combinatorios. Este es inspirado por las habilidades de las hormigas reales para encontrar los senderos más pequeños entre su nido y un recurso de comida.

Mientras caminan desde las fuentes de comidas hasta sus nidos y viceversa, las hormigas depositan una substancia llamada feromona sobre el suelo. Cuando ellas deciden acerca de la dirección a donde ir, ellas escogen con probabilidad más alta los senderos que contienen concentraciones de feromonas más fuertes. Este comportamiento básico es la base para una interacción cooperativa la cual resulta en el descubrimiento de los senderos más cortos.

Los algoritmos ACO están basados sobre un modelo probabilístico parametrizado --el modelo de feromona – que se utiliza para modelar los caminos de feromonas químicas. Las hormigas artificiales incrementalmente construyen soluciones

¹⁰ Algoritmos meta heurísticos: ver glosario

añadiendo oportunamente componentes de la solución definida a una solución parcial bajo consideración.

Para hacer esto, las hormigas artificiales realizan paseos al azar en un gráfico completamente conectado $G=(C, L)$, cuyos vértices son: los componentes de la solución C y el conjunto L son las conexiones.

El gráfico es comúnmente llamado Gráfico de Construcción.

Cuando una restricción de problemas de optimización combinatoria es considerada, las restricciones del problema se construyen en el procedimiento constructivo de las hormigas de tal manera que en cada paso del proceso de construcción solo los componentes de la solución viable pueden ser añadidos a la solución parcial actual. En la mayoría de aplicaciones, las hormigas son implementadas para construir soluciones viables, pero algunas veces también es deseable permitirles. El valor de cada parámetro de senderos de feromona es usualmente denotado por T_i . El conjunto de todos los parámetros de senderos de feromonas es denotado por T . Estos valores de feromonas son usados por las hormigas para tomar decisiones probabilísticas de cómo moverse sobre el gráfico de construcción. Más específicamente, una hormiga es un agente computacional simple, la cual construye una solución para la instancia a resolver.

1.4.1. ACO INCLUYE DOS MECANISMOS

1.4.1.1. Sendero de Evaporación

Reduce todos valores del sendero con el tiempo, con el fin de evitar la acumulación ilimitada de senderos sobre algunos componentes.

1.4.1.2. Acciones Demonio

Pueden ser usadas para implementar acciones centralizadas las cuales no pueden ser desempeñadas por hormigas únicas. Por ejemplo, el uso de un procedimiento de búsqueda local aplicado a las soluciones construidas por las hormigas, o la colección de información global que puede ser usada para decidir si es útil o no depositar feromona adicional para sesgar el proceso de búsqueda desde una perspectiva no local.

Como un ejemplo práctico, el demonio puede observar el sendero encontrado por cada hormiga en la colonia y elegir depositar feromona extra sobre los

componentes usados por la hormiga que construye la mejor solución. El desempeño en la actualización de feromonas por el demonio son llamadas actualizaciones de feromonas fuera de línea.

1.4.2. ALGORITMOS BASADOS EN ACO

1.4.2.1. ANT SYSTEM (AS)

La importancia del algoritmo Ant System reside principalmente en ser el prototipo de un número de algoritmos de hormigas las cuales implementan colectivamente el paradigma ACO.

En el algoritmo Ant System se construyen las soluciones de la siguiente forma: para cada hormiga k en cada paso de construcción se escoge ir del nodo i al siguiente nodo j . $\forall j$, con una probabilidad P_{ij}^k

$$P_{ij}^k = \frac{(\tau_{ij})^\alpha * (n_{ij})^\beta}{\sum_{j \in N_i^k} (\tau_{ij})^\alpha * (n_{ij})^\beta} \quad \text{si } j \in N_i^k$$

donde N_i^k es el vecindario alcanzable por la hormiga k cuando se encuentra en el nodo i ; α es el factor de escalado de feromona y β el de visibilidad, ambos se usan para afinar el proceso de búsqueda; τ_{ij} el valor de feromona en el arco que une los nodos i y j ; n_{ij} se denomina función de visibilidad, que depende totalmente de las características del problema que se va a resolver, por ejemplo para el TSP es $\frac{1}{d_{ij}}$ donde d_{ij} es la distancia entre las ciudades i y j . Luego se comparan para todas las hormigas sus soluciones encontradas con la mejor hasta el momento y se modifica este si alguna de las encontradas la mejora. En la actualización de la huella de la feromona se evapora una proporción constante de feromona en cada arco y luego cada hormiga una vez que la solución está completa deposita una cantidad de feromona en dependencia de la calidad de su solución, o sea, actualización en línea a posteriori.

1.4.2.2. ANT COLONY SYSTEM (ACS)

Ant System fue el primer algoritmo inspirado por el comportamiento de hormigas reales. Ant System fue inicialmente aplicado a la solución del problema del vendedor viajero pero no fue capaz de competir contra el estado de los algoritmos

en el campo. Por otro lado tiene el mérito de introducir algoritmos ACO y mostrar la potencialidad de usar feromona artificial y hormigas artificiales para conducir la búsqueda de mejores soluciones para problemas de optimización complejos. La siguiente investigación fue motivada por dos objetivos: la primera fue para mejorar el rendimiento del algoritmo y el segundo fue para investigar y explicar mejor su comportamiento. Gambardella y Dorigo proponen en 1995 el algoritmo Ant-Q, una extensión de Ant System la cual integra algunas ideas de Q-learning¹¹, y en 1996 Ant Colony System una versión simplificada de Ant-Q¹² la cual mantiene aproximadamente el mismo nivel de rendimiento, medido por la complejidad del algoritmo y por resultados computacionales. Desde entonces ACS es la base de muchos algoritmos definidos en los siguientes años. ACS es diferente de AS por dos principales aspectos:

Feromona

En ACS una vez que todas las hormigas han calculado su viaje (al final de cada iteración) AS actualiza el rastro de la feromona usando todas las soluciones producidas por la colonia de hormigas. Cada borde perteneciente a una de las soluciones calculadas es modificado por una cantidad de feromona proporcional a su valor de solución. Al final de esta fase la feromona de los sistemas enteros se evapora y el proceso de construcción y actualización es iterado. Por el contrario, en ACS solo la mejor solución calculada desde el principio de la computación es usada para *actualizar globalmente* la feromona. Como fue el caso en AS, la actualización global está intentando incrementar el atractivo de una ruta prometedora pero el mecanismo ACS es más efectivo desde que este evita tiempos de convergencia prolongados para directamente concentrar la búsqueda en una vecindad del mejor circuito encontrado hasta la iteración actual del algoritmo.

En ACS, la fase final de evaporación es substituida por una actualización local de la feromona aplicada durante la fase de construcción.

¹¹ Q-learning es un tipo de algoritmo de aprendizaje a través del reforzamiento de conductas o patrones deseables.

¹² Ant-Q es una adaptación del algoritmo Ant System de manera que las hormigas artificiales aprendan a través de los rastros de feromona usando el reforzamiento conductual de Q-learning.

Regla de Transición de Estado

Durante la construcción de una nueva solución la regla de transición de estado es la fase donde cada hormiga decide cual es el siguiente estado para moverse. En ACS una nueva regla de transición de estado llamada *pseudo-aleatorio-proporcional* es introducida. La regla *pseudo-aleatorio-proporcional* es un compromiso entre la regla de elección de estado *pseudo-aleatorio* típicamente usada en Q-learning y la regla de elección de acción *aleatorio-proporcional* típicamente usada en Ant System. Con la regla *pseudo-aleatorio* el estado de elección es el mejor con probabilidad q_0 (explotación) mientras que un estado aleatorio es elegido con probabilidad $1 - q_0$ (exploración). Usando la regla *aleatorio-proporcional* del AS el siguiente estado es elegido aleatoriamente con una distribución de probabilidad en función de n_{ij} y τ_{ij} . La regla de transición de estado del ACS *pseudo-aleatorio-proporcional* provee una manera directa de equilibrio entre exploración de nuevos estados y explotación de un a priori y conocimiento acumulado.

1.4.2.3. MIN-MAX ANT SYSTEM (MMAS)

Introduce cuatro principales modificaciones con respecto a Ant System. Primero, explota fuertemente las mejores rutas encontradas: solamente la *mejor- iteración* de hormigas, que es, la hormiga que produce el mejor viaje en la iteración actual, o la hormiga que a producido el mejor viaje conocido hasta el momento es permitida para depositar feromona. Desafortunadamente, tal estrategia puede llevar a una situación de estancamiento en la cual todas las hormigas siguen el mismo viaje, debido al crecimiento excesivo de rastros de feromona en los arcos de un buen tour, aunque este sea subóptimo. Para contrarrestar este efecto, una segunda modificación introducida por MMAS es que este limite el posible rango de valores de senderos de feromonas para el intervalo (T_{\min}, T_{\max}) mostrados a continuación. Tercero, los senderos de feromonas son inicializados a los limites superiores de senderos de feromonas, el cual, junto con una pequeña proporción de evaporación de feromona, incrementa la exploración de viajes al principio de la búsqueda. Finalmente, en MMAS, los senderos de feromonas son reinicializados cada vez que los sistemas se enfocan al estancamiento o cuando ningún viaje

mejorado ha sido generado por un cierto número de iteraciones consecutivas. El Max-Min Ant System introduce límites altos y bajos sobre el valor de feromonas. Si la diferencia entre algunos valores de feromonas fueron también amplias, todas las hormigas generarían casi siempre las mismas soluciones, los cuales significarían estancamiento de algoritmos. Los límites sobre los valores de feromona previenen eso. La diferencia máxima entre los niveles más altos y más bajos de feromona pueden ser controlados, y así el nivel de intensificación de búsqueda versus diversificación puede ser balanceado. La regla de actualización de feromona se convierte entonces en la siguiente: (para el caso particular de asignación de eventos e en intervalos de tiempo t):

$$T(e,t) \leftarrow \begin{cases} (1-p) \cdot T(e,t) + 1 & \text{if } (e,t) \text{ es en } C_{global_best} \\ (1-p) \cdot T(e,t) & \end{cases}$$

Donde $p \in [0,1]$ es la cantidad de evaporación. La feromona actualizada es completada usando lo siguiente:

$$T(e,t) \leftarrow \begin{cases} T_{min} & \text{if } T(e,t) < T_{min}, \\ T_{max} & \text{if } T(e,t) > T_{max}, \\ T(e,t) & \end{cases}$$

Parámetros usados por los algoritmos

Nombre Parámetro	MMAS	ACS
M	Número de hormigas	
P	Evaporación de feromonas	
s(j)	Número de pasos de búsqueda local	
τ_0	Valor con el cual la matriz de feromona es inicializada	
T_{max}	Nivel máximo de feromona	-
T_{min}	Nivel mínimo de feromona	-
α	-	Feromona local
β	-	Peso de restricciones difíciles
γ	-	Peso de restricciones indemostrables
G	-	Adaptando el factor

Tabla 1.1 Parámetros usados por los algoritmos¹³

¹³ Tomado Ant Algorithms de Krzysztof Socha, Michael Samples and Max Manfrin, 2000

1.4.2.4. APPROXIMATE NONDETERMINISTIC TREE SEARCH (ANTS)

ANTS es un algoritmo ACO que explota ideas de programación matemática. En particular, ANTS calcula límites inferiores a la finalización de una solución parcial para definir de la información heurística que es usada por cada hormiga durante la construcción de la solución. El nombre ANTS se deriva del hecho de que el algoritmo propuesto pueden ser interpretado como una búsqueda de árbol no determinística aproximada desde entonces esto puede ser extendido en un sencillo camino para una rama y un procedimiento limitado. De hecho, el algoritmo ANTS es extendido a un algoritmo exacto. Aparte del uso de límites más bajos, ANTS también introduce 2 modificaciones adicionales con respecto a AS: El uso de una nueva regla de elección de acción y una modificación en la regla de actualización de sendero de feromonas

Construcción de la Solución

La regla usada por ANTS para calcular las probabilidades durante la construcción de la solución de las hormigas tiene una forma diferente a la que se usa en otros algoritmos ACO. En ANTS, una hormiga k que está situada en la ciudad i elige la siguiente ciudad j con una probabilidad dada por:

$$p_{ij}^k = \frac{\zeta T_{ij} + (1 - \zeta) n_{ij}}{\sum_{j \in N_j^k} \zeta T_{ij} + (1 - \zeta) n_{ij}}, \quad \text{if } j \in N_i^k$$

donde ζ es un parámetro, $0 \leq \zeta \leq 1$, y N_j^k es la vecindad factible (como de costumbre, la probabilidad de elegir un arco no perteneciente N_j^k es 0).

1.4.2.4.1. Características de ANTS

- Usa ideas no incluidas en el AS original.
- No aplicado para TSP.
- Crea heurísticas dinámicas

2. CAPÍTULO 2. DESARROLLO DEL MÉTODO

2.1. CARACTERIZACIÓN DEL PROBLEMA

El problema de calendarización de clases en el ámbito universitario básicamente consiste en, dados: un conjunto de semestres o grupos de materias con sus respectivos paralelos, un conjunto de profesores, un conjunto de periodos de tiempo disponibles cada día de la semana, y un conjunto de aulas; en asignar los paralelos de las materias a los profesores que dictaran las clases, estos mismos paralelos a los periodos de tiempo dentro de los días de la semana en los cuales las clases serán dictadas, y a las aulas donde las clases se llevaran a cabo; todo esto sujeto a un conjunto de restricciones suaves y duras. La mayor dificultad estriba en encontrar soluciones al problema que estén libres de conflicto es decir, estas no deben violar ninguna de las restricciones duras.

Debido a estas características se ha incluido al problema dentro de la categoría de problemas de optimización combinatoria, altamente restringidos. Por estos motivos no siempre es posible dar una solución que sea del todo aceptable para todas las partes involucradas.

La amplia variación de la cultura organizativa, objetivos competitivos, un conjunto cambiante de restricciones y políticas aplicadas dentro de cada una de las instituciones de educación superior a nivel mundial hacen del problema difícil de colocar dentro de un contexto general y que la elaboración de un modelo que abarque la realidad de todas sea en la práctica casi imposible de realizar.

Lo que en esta sección se mostrará de manera rápida es una breve visión de las características más comunes que podría contener un modelo general.

Desde hace algunos años una organización europea conocida como PATAT (Practice and Theory of Automated Timetabling) ¹⁴ ha venido llevando a cabo el ITC (International Timetabling Competition), con el objetivo de incentivar la investigación y el diseño de software y algoritmos que traten con la calendarización automatizada.

Dentro de las secciones del concurso, existe una dedicada exclusivamente al problema de calendarización de clases a nivel universitario (Course Timetabling),

¹⁴ <http://www.cs.qub.ac.uk/~B.McCollum/patat10/>

a su vez esta se divide en dos categorías: el modelamiento basado en el currículo (Curriculum Based), y el modelamiento tomando en cuenta a los estudiantes una vez que estos ya se han inscrito en las respectivas materias (Post Enrollment Based). Nos parece importante explicar de manera sucinta, la categorización efectuada dentro del seno de la ITC, ya que son dos de los enfoques más comúnmente usados dentro de la literatura concerniente al tema.

2.1.1. Modelamiento Basado en el Currículo:

El problema de calendarización de horarios basado en el currículo consiste en organizar un horario semanal, para las clases de las diferentes materias que ofrece la universidad, dado un número de aulas y periodos de tiempo disponibles. Aquí los conflictos en la calendarización de las diferentes materias surgen por cómo están situadas dentro del currículo (si están relacionadas como prerrequisito unas con otras, o pertenecen al mismo grupo dentro del currículo, es decir deben tomarse a un mismo tiempo) y no por los datos de la inscripción de los alumnos.

2.1.2. Modelamiento Basado en los Datos de Inscripción:

Luego de que los estudiantes, se inscriben en las diferentes materias que van a tomar, se construye un horario de tal manera que se garantice que la mayoría de los estudiantes puedan asistir a todas las clases de las materias para las cuales se inscribieron. Es decir se busca garantizar que la mayoría de estudiantes no tenga cruces de horario.

2.2. DEFINICIÓN DE LOS TIPOS DE RESTRICCIONES

Los requerimientos de cada institución como ya se ha mencionado antes, son diversos; por esta razón y dado que las restricciones dependen directamente de los requerimientos, hay un conjunto muy amplio de estos relacionados con el problema de calendarización a nivel universitario. En cada institución existe además un conjunto de actores cada uno con diferentes prioridades e intereses, afectados y beneficiados por los resultados de la calendarización. Existen tres principales categorías de personas afectadas por los resultados de este proceso:

los personeros de la administración, los estudiantes y los profesores. Consecuentemente la calidad de un horario puede ser valorada desde diferentes puntos de vista, dependiendo de los intereses de cada grupo de interés.

Loo y otros [1986], han propuesto 4 categorías generales para las restricciones: Requerimientos de Espacio, de Paralelos, de la Administración, y de la Clase.

Entre los requerimientos de espacio tenemos:

- El uso del aula debe ser maximizado.
- Las aulas que tiene preferencia para cada semestre deben ser usadas primero.
- Ciertas aulas deben ser usadas tanto como sea posible (Laboratorios, salas de uso múltiple, auditorios, etc.).
- Ciertas aulas (por ejemplo las salas de conferencias) no deben ser usadas a menos de que no quede ninguna otra opción.
- Ciertas aulas deben usarse para alguna materia en específico.
- Debería ser posible agrupar aulas de acuerdo a la funcionalidad que posean.
- La carga de estudio para los alumnos debe ser distribuida equitativamente a lo largo de la semana.
- Algunos paralelos pueden requerir un periodo común de tiempo libre una vez a la semana para trabajar en sus proyectos.
- Las horas consecutivas para clases o tutorías, no debe exceder un tamaño predefinido de tiempo.
- Debe haber límites en el número de clases y tutorías en los que participarán los estudiantes cada día.
- Deben proveerse pausas para el almuerzo y la cena a los estudiantes.
- Cada paralelo debe tener suficientes oportunidades para asistir a la mayoría de las materias opcionales.
- Los estudiantes deben ser capaces de ser divididos en grupos de tutorías o combinados para conferencias en común.
- Debería haber un límite en el número total de eventos de primer período a la que asistieron los estudiantes cada semana.

- La carga de trabajo para los profesores debe ser distribuida uniformemente a lo largo de la semana.
- Las limitaciones de tiempo para los profesores de tiempo parcial deben ser tomadas en cuenta.
- Los profesores de tiempo completo pueden requerir algunos periodos de tiempo libre a la semana para asistir a reuniones.
- Las horas consecutivas al día de clases o tutorías de un profesor no deben superar cierto valor predeterminado.
- Deben existir límites al número de horas al día que un profesor dicta clases u ofrecer tutorías.
- Los profesores deben poseer pausas para el almuerzo o la cena, estas pausas pueden ser escalonadas.
- Se debe permitir que diferentes maestros dicten diferentes aspectos de una materia.
- Deben existir límites al número total de clases en el primer periodo que dicta un profesor.
- Debería dejarse libre la mañana para profesores o estudiante que tengan eventos en la tarde ese mismo día
- Clases repetidas (con gran concurrencia) no deben ser impartidas por el mismo profesor varias veces al día.
- Estudiantes de diferentes aéreas deben poder ser agrupados para clases en común de ser necesario.
- Es preferible que las clases y tutorías se lleven a cabo en la mañana de ser posible.
- El sábado debería ser un día libre tanto para los estudiantes como para los profesores, si esto es posible.
- A veces es posible tener un descanso de al menos 15 minutos entre clases o tutorías.
- Algunos eventos especiales deben ser calendarizados manualmente.

Ven y otros [1994] describen 5 tipos diferentes de restricciones de carácter general que pueden aplicarse a los problemas de calendarización universitaria.

2.2.1. Restricciones Unitarias

Que involucran un solo evento; estas caen en dos categorías:

Exclusiones: Un evento no puede llevarse a cabo en una determinada aula o a hora en particular, o no puede ser asignado a determinado agente (por ejemplo un profesor de tiempo parcial únicamente dicta clases en las horas que tiene disponibles, por tanto los paralelos que están a su cargo solo pueden recibir clases dentro de estos tiempos).

Especificaciones: Un evento debe llevarse a cabo en una determinada aula, o a hora en particular, o debe ser asignado a determinado agente (por ejemplo una asamblea general solo puede llevarse a cabo en una auditorio y en ningún otro tipo de aula).

2.2.2. Restricciones Binarias

Que involucran al menos dos eventos, caen en dos categorías.

Restricciones de Borde: Surgen por el simple hecho de que las personas no pueden estar en dos lugares al mismo tiempo (por ejemplo un profesor no puede dictar dos clases al mismo tiempo en dos lugares diferentes).

Restricciones Yuxtaposición: El orden o los espacios de tiempo entre dos eventos están restringidos de alguna manera (por ejemplo existe un número máximo de “horas huecas” dentro del horario).

2.2.3. Restricciones de Capacidad

Especifican que alguna función de un conjunto dado de eventos que ocurren simultáneamente en un lugar determinado no pueden exceder un máximo dado (por ejemplo un laboratorio que tiene capacidad para 20 personas no debe recibir grupos de estudiantes que excedan esta capacidad).

2.2.4. Restricciones de repartición de eventos

La manera en que los eventos se distribuyen en el tiempo. Puede requerir que múltiples clases del mismo tópico, deban ser distribuidos en el tiempo tan uniformemente como sea posible, durante la semana (por ejemplo se podría requerir que las clases de cierto paralelo de cierta materia se lleven a cabo los lunes, miércoles y viernes, sin perjuicio de la hora).

2.2.5. Restricciones de Agente

Involucran restricciones al tiempo total asignado a un agente dentro de horario, y restricciones o especificaciones sobre los eventos en que cada agente individual puede estar involucrado (por ejemplo los profesores de tiempo completo deben dictar al menos 20 horas de clases a la semana).

La clasificación realizada en los párrafos anteriores esta efectuada desde el punto de vista del problema, sin embargo, es posible también realizar una clasificación que tenga relación con la obligatoriedad de respetar las restricciones:

2.2.6. Restricciones Duras

Son restricciones que expresan limitaciones absolutas dentro de los modelos, es decir deben ser cumplidas en todos los casos sin excepción. Estas limitan el espacio de solución, dado que cualquier asignación de las variables que no las cumpla se considera como no viable, y no pertenece al espacio de solución.

2.2.7. Restricciones Suaves

Representan preferencias en el modelo. Cada asignación de las variables es tomada en cuenta, a través de una función (objetivo) que fija su nivel de deseabilidad. Dado que estas están basadas en medidas cuantitativas, son muy expresivas en cuanto a la deseabilidad de una solución en particular.

La clasificación mostrada a continuación muestra el nivel de obligatoriedad de las restricciones mas usualmente encontradas dentro de la literatura existente sobre el tema:

La mayoría de investigadores coinciden en el siguiente conjunto de restricciones duras para el problema de calendarización universitaria:

- Todas materias deben estar asignadas a algún profesor.
- Todos los profesores deben estar asignados para dictar clases de alguna materia.
- Ningún profesor debe estar asignado para dictar más de una clases al mismo tiempo.

- Ningún profesor debe estar asignado para dictar clases en horas en la que no está disponible.
- El requisito de horas mínimas de trabajo para un profesor debe llevarse a cabo.
- El número de horas de tiempo extra para un profesor no debe ser mayor en ningún caso, al especificado.
- Ningún estudiante o paralelo debe ser asignado para recibir más de una clase al mismo tiempo.
- Una clase debe ser asignada a al menos un periodo.
- Dos clases de materias de un mismo semestre no pueden ser calendarizadas al mismo tiempo.
- Clases de materias etiquetadas como de la mañana deben llevarse a cabo en la mañana.
- Clases de materias etiquetadas como de la tarde deben llevarse a cabo en la tarde.
- Clases o eventos pre asignados, deben ser calendarizados en los periodos pre asignados.
- Algunos eventos (clases) se limitan a determinados horarios y no deben ser asignados.
- Algunos eventos (clases) necesitan ser celebrados consecutivamente.
- Cada aula no puede contener más de un evento en cualquier periodo dado.
- La capacidad de las aulas asignadas a los eventos (clases) deben ser igual o mayor que los estudiantes.
- Un evento solo puede asignarse a una clase si y solo si la habitación está disponible.
- Las aulas pre asignadas pueden ser calendarizadas en las aulas pre asignadas.
- Algunas clases son limitadas para alguna aula en específico y pueden no ser asignadas.
- Ciertas clases pueden ser enseñadas en la misma aula y al mismo tiempo.
- Ciertas clases deberían tener exactamente el mismo tiempo de enseñanza pero diferentes aulas.

La mayoría de investigadores coinciden en el siguiente conjunto de restricciones suaves para el problema de calendarización universitaria:

- Un profesor no debería enseñar más de un número especificado del máximo semanal de carga docente.
- Un profesor no debería enseñar más de un número especificado del máximo diario de carga docente.
- El profesor debe tener al menos un tiempo libre entre dos tiempos de enseñanza.
- Algunos profesores, por el contrario, desean tener todas sus clases calendarizadas para períodos consecutivos.
- Los docentes deben ser asignados a las clases de su preferencia/cursos.
- Un evento no debe ser asignado a un determinado profesor.
- La mañana debe ser libre para profesores que tengan clases en la tarde del mismo día.
- Ningún estudiante o grupo de estudiantes puede atender más que x clases en un día.
- Un estudiante no debe tener solo una clase en un día.
- Un estudiante o grupo de estudiantes no les gusta tener mucho tiempo libre entre dos clases.
- Eventos de cursos opcionales de un programa de estudio debe ser calendarizado en menos tiempo.
- Clases del mismo curso y grupo de estudiantes no deben ser calendarizados en el mismo día.
- Un grupo de estudiantes con la misma clase y programa de estudio deben ser calendarizados en el mismo horario.
- Un grupo de estudiantes con la misma clase pero diferente programa de estudio no deben ser calendarizados al mismo tiempo.
- Si una clase se lleva a cabo más de una vez a la semana siempre habrá por lo menos un día en medio.
- Los eventos no deberían ser asignados durante el tiempo del lunch.

- Cierta grupo de estudiantes pueden requerir un tiempo libre una vez a la semana por sus proyectos de trabajo.
- Las clases deben ser asignadas a las aulas de tal manera que el aula pueda ser utilizada al máximo.
- Los eventos deben ser calendarizados en las aulas lo más cerca posible de su profesor.
- Todas las clases del mismo grupo de estudiantes deben ser calendarizadas en la misma aula.
- Debe haber un límite en el número total de aulas asignadas al primer horario de clases.
- Las aulas no deben ser calendarizadas durante el tiempo en que estas han sido reservadas.
- Los estudiantes o grupo de estudiantes deben ser asignados a las aulas de su preferencia.
- Suficiente tiempo debe ser proveído para que los estudiantes se movilicen de un aula a otra.

2.3. MODELAMIENTO MATEMÁTICO DEL PROBLEMA

El modelo matemático que se presenta a continuación está basado en la información referente al problema de calendarización en el ámbito universitario en general, aunque no todas las restricciones tomadas en cuenta en las secciones anteriores se encuentran presentes, sino, solo las que se han encontrado más relevantes. Este es un primer paso para ir de lo general hacia lo particular, lo que se hará con más énfasis cuando se realice el análisis para el caso de estudio.

2.3.1. CONJUNTO DE DATOS:

Las entidades matemáticas a continuación descritas están representadas en forma de vectores de estructuras de datos.

Materia:

Representa a cada una de las materias que la universidad imparte respecto a un área del conocimiento determinada. El dominio de esta variable **M**, es el conjunto de las todas materias que se dictan en todos los semestres, en los que está dividida una carrera en la universidad. Cada materia $m_i \in \{m_1, m_2, m_3 \dots m_{n_1}\}$ está asociada con un número de paralelos N_i y con un número de créditos es decir el número de horas/clase que son necesarias dictar para cubrir los temas incluidos en la materia. Existen dos tipos de materias opcionales y obligatorias. Una materia obligatoria es aquella que se debe aprobar para completar la carrera. Una materia opcional no necesariamente se debe aprobar para completar la carrera, pero es necesario aprobar un conjunto mínimo de estas para completar la carrera sin discriminar individualmente cada una de estas.

(n_1 Representa el número total de materias)

Paralelo:

Cada paralelo $p_j \in \{p_1, p_2, p_3 \dots p_{n_2}\}$ representa un grupo de alumnos que toman una materia m_i (se dice que el paralelo está asociado a la materia), está dirigido por un profesor t_j , posee un conjunto semanal de clases que se dictaran dentro del rango horario asociado con el paralelo. Se establece también un número mínimo y máximo de alumnos dentro del paralelo y tiene asociado un conjunto de aulas. El dominio de esta variable **P**, es el conjunto de todos los grupos de estudiantes, que toman materias en cualquiera de los semestres.

(n_2 Representa el número total de paralelos)

Aula:

Representa una de las aulas disponibles dentro de las instalaciones de la universidad, donde se dictaran las clases. El dominio de esta variable **A**, es el conjunto de todas las aulas disponibles en la universidad. Cada aula $a_i \in \{a_1, a_2, a_3 \dots a_{n_3}\}$ tiene asociado un tipo (normal, laboratorio, sala de conferencias), y una capacidad, esta nos dice el número máximo de estudiantes que sería recomendable asistan simultáneamente a una clase dictada en el aula.

(n_3 Representa el número total de aulas)

Semestre:

Representa un grupo de materias, cada semestre $s_i \in \{s_1, s_2, s_3 \dots s_{n_4}\}$ tiene asociado un conjunto de aulas predefinidas y un grupo de materias obligatorias y opcionales. Además por conveniencia se añade un tipo asociado a cada semestre: de la mañana o de la tarde. El dominio de esta variable **S**, es el conjunto de todos los semestres de todas las carreras de la universidad.

(n_4 Representa el número total de semestres)

Clase:

Una clase $c_i \in \{c_1, c_2, c_3 \dots c_{n_5}\}$ representa una disertación dictada a un paralelo p_j que pertenece a una materia m_k , por un profesor t_l , en un aula a_n en un horario determinado. Cada clase está asociada a una duración esto es el número de periodos que esta necesita para llevarse a cabo y el tipo de aula que esta requiere. El dominio de esta variable **C**, es el conjunto de todas las clases dictadas a todos los paralelos de todas las materias por todos los profesores.

(n_5 Representa el número total de clases)

Período:

Representa una hora en un día de la semana en la que se dictaran clases, El dominio de esta variable **H**, es el conjunto de todos los periodos validos dentro de la semana. Cada periodo $h_i \in \{h_1, h_2, h_3, \dots, h_{n_6}\}$ tiene asociadas una hora **f** y un día **d**; es decir es un par de la forma **(d,f)**, la hora **f** tiene un formato de 24 horas.

(n_6 Representa el número total de periodos)

Rango Horario:

Representa un conjunto de periodos continuos, es decir que la diferencia de sus horas sea 1. Se usan para establecer una preferencia horaria de los paralelos, es decir en que rango de periodos se determinara que sus clases van a ser dictadas. El dominio de estas variable es **U**, y es el conjunto de todos los rangos horarios definidos en el problema.

(n_9 Representa el número total de rangos horarios)

Profesor:

Representa un disertador, que imparte sus conocimientos a los alumnos de la universidad. El dominio de esta variable **T**, es igual al conjunto de todos los profesores que trabajan para la universidad. Cada profesor $t_j \in \{t_1, t_2, t_3 \dots t_{n_7}\}$ está asociado a un conjunto de materias que puede dictar, cada una con un nivel de preferencia, un conjunto de los periodos en que está disponible, un tipo (tiempo completo o tiempo parcial) y un número de horas mínimo en las que debe dictar clases durante la semana.

(n_7 Representa el número total de profesores)

Estudiante:

Representa un estudiante matriculado para tomar las clases de al menos alguna materia perteneciente a un semestre. El dominio de esta variable **E**, es igual al conjunto de todos los estudiantes matriculados en la universidad. Cada estudiante $e_i \in \{e_1, e_2, e_3 \dots e_{n_8}\}$, está asociado con un conjunto de materias en la cuales se halla matriculado.

(n_8 Representa el número total de estudiantes)

2.3.2. MATRICES DE ENTRADA:

Las entidades matemáticas descritas a continuación son matrices binarias codificadas de la siguiente manera:

Por ejemplo, tomando 3 semestres y 4 materias; si en el semestre 2 se dicta la materia 4 y en el semestre 1 se dicta la materia 3 la matriz binaria semestre-materia **I** lucirá como se muestra en la figura 2.1

		Materias			
		1	2	3	4
Semestres	1	0	0	1	0
	2	0	0	0	1
	3	0	0	0	0

Figura 2.1 Matriz Binaria Semestre-Materia

La Notación usada es la representación del elemento general $elemento_{ij}$ de las matrices.

Paralelo-Materia: $G=P \times M$, $g_{ij} = 1$ si el paralelo i está asociado con la materia j , 0 caso contrario; $N_g(m_j)$ es el numero de paralelos asociados con la materia j .

Paralelo-Clase: $B = P \times C$, $b_{ij} = 1$ si la clase j es dictada al paralelo i , 0 caso contrario; $N_b(p_i)$ es el numero de clases dictadas al paralelo i y $N_b(c_j) = 1$ ya que una clase en particular solo puede ser dictada a uno y solo un paralelo.

Profesor-Materia (preferencia): $E=T \times M$, e_{ij} representa el nivel de preferencia del profesor para dictar una materia, así: 1 representa el valor más alto, 2 representa el segundo valor más alto, etc.

Esta es una excepción la matriz aquí denota si el profesor puede dictar una materia o no y si lo puede hacer, nos muestra el grado de preferencia que el profesor tiene al dictar clases en las diferentes materias. En el grafico se describe de manera gráfica esta situación:

Nil o null quiere decir que el profesor no puede dictar clases de las materias marcadas así:

		Materia			
		1	2	3	4
Profesor	1	nil	1	2	6
	2	1	nil	2	1
	3	1	3	nil	2

Figura 2.2 Nivel de Preferencia del Profesor para dictar una materia

Periodo-Profesor: $F=H \times T$, $q_{ij} = 1$ si el profesor i está disponible durante el periodo j , 0 caso contrario; $N_f(t_i)$ es el número de periodos en los que está disponible el profesor i y $N_f(h_j)$ es el número de profesores que están disponibles en el periodo j .

Semestre-Materia: $I= S \times M$, $i_{ij} = 1$ si el semestre i contiene a la materia j , 0 caso contrario; $N_i(s_i)$ es el número de materias asociadas con el semestre i .

Semestre-Aula (preferencia): $R = S \times A$, $r_{ij} = 1$ si el semestre i tiene una preferencia superior a la media para que las clases de los paralelos de las materias que los conforman sean asignadas al aula j ; $N_R(s_i)$ es el número de aulas que están asignadas al semestre i para recibir sus clases; $N_R(a_i)$ es el número de semestres que tienen una preferencia más alta para que sus clases se realicen en el aula j .

Aula-Periodo (restricción): $O = A \times P$, $o_{ij} = 1$ si el aula i está disponible en el periodo j , es igual a 0 caso contrario

Materia-Materia: $Z = M \times M$, $z_{ij} = 1$ si la materia i , no está concatenada con la materia j , es igual a 0 caso contrario;

2.3.3. MATRICES DE SALIDA:

Paralelo-Profesor: $L = P \times T$, $l_{ij} = 1$ si al paralelo i le ha sido asignado el profesor j , 0 caso contrario; $N_l(p_i) = 1$ es el número de profesores que dictan clases en el paralelo i y $N_l(t_j)$ es el número de paralelos en los que dicta clases el profesor j .

Periodo-Clase-: $K = H \times C$, $k_{ij} = 1$ si durante el periodo i ha sido programada para desarrollarse en la clase j , 0 caso contrario; $N_k(h_i)$ es el número de clases dictadas en el periodo i y $N_k(c_j)$ es el número de periodos en los que se dicta la clase j .

Clase-Aula: $Y = C \times A$, $y_{ij} = 1$ si la clase i tiene asignada el aula j en algún periodo dado; $N_Y(c_i)$ es el número de clases que están asignadas al aula j ;

Paralelo-Estudiante: $W = P \times E$, $w_{ij} = 1$ si el estudiante j recibirá clases en el paralelo i , 0 caso contrario;

2.3.4. RESTRICCIONES DURAS:

Profesores:

[H1] Un profesor debe estar asignado a por lo menos un paralelo:

$$\forall i \in \{1 \dots n_7\}, \sum_{j=1}^{n_2} l_{ij} \geq 1$$

[H2] Un paralelo debe estar asignado al menos a un profesor:

$$\forall j \in \{1 \dots n_2\}, \sum_{i=1}^{n_7} l_{ij} \geq 1$$

[H3] Ningún profesor puede dar dos clases al mismo tiempo:

Primero, definimos la función auxiliar de los cálculos definimos la función:

$$\theta(p_n, p_m) = \sum_{i=1}^{n_7} l_{in} l_{im}$$

Por tanto,

$$\theta(p_n, p_m) = \begin{cases} 1 & \text{si los paralelos, } n \text{ y } m \text{ comparten al menos un mismo profesor} \\ 0 & \text{caso contrario} \end{cases}$$

Definimos además la función auxiliar,

$$\phi(p_i, h_o) = \sum_{j=1}^{n_5} b_{ij} k_{oj}$$

Por tanto:

$$\phi(p_i, h_o) = \begin{cases} 1 & \text{si el paralelo } i \text{ tiene alguna de sus clases calendarizada en el periodo } o \\ 0 & \text{caso contrario} \end{cases}$$

Por tanto la restricción puede definirse de la siguiente manera:

$$\sum_{n=1}^{n_2-1} \sum_{m=n+1}^{n_2} \sum_{o=1}^{n_6} \phi(p_n, h_o) \phi(p_m, h_o) \theta(p_n, p_m) = 0$$

[H4] El profesor debe tener clases solo en los periodos en que está disponible.

$$l_{ki} \phi(p_k, h_j) \leq q_{ij}, \forall i \in \{1 \dots n_7\}$$

[H5] Carga mínima de trabajo de los profesores.

Definimos la función:

$N_h(c_i)$ Cuyo valor es el número de periodos que la clase c_i toma para llevarse a cabo.

Definimos además la función:

$H_t(t_j)$ Cuyo valor es el número de periodos mínimos que debe laboral el profesor t_j en la semana.

Por tanto; debe cumplirse que:

$$\forall j \in \{1 \dots n_7\} \sum_{i=1}^{n_2} \sum_{k=1}^{n_5} l_{ij} b_{ik} N_k(c_k) \geq H_t(t_j)$$

[H6] Carga máxima de trabajo de los profesores.

Definimos además la función:

$H_m(t_j)$ Cuyo valor es el número máximo de periodos que debe laboral el profesor t_j en la semana.

Debe cumplirse que:

$$\forall j \in \{1 \dots n_7\} \sum_{i=1}^{n_2} \sum_{k=1}^{n_5} l_{ij} b_{ik} N_k(c_k) \leq H_m(t_j)$$

Clases:

[H7] El mismo estudiante no puede asistir a dos clases al mismo tiempo.

Definimos la función auxiliar

$$\psi(p_i, p_n, e_j) = w_{ij} w_{nj}$$

De modo que:

$$\psi(p_i, p_n, e_j) = \begin{cases} 1, & \text{si el estudiante } j \text{ esta inscrito en los paralelos } i \text{ y } n \\ 0 & \text{caso contrario} \end{cases}$$

$$\sum_{j=1}^{n_8} \sum_{i=1}^{n_2-1} \sum_{n=i+1}^{n_2} \sum_{m=1}^{n_6} \phi(p_i, h_m) \phi(p_n, h_m) \psi(p_i, p_n, e_j) = 0$$

[H8] Dos paralelos del mismo semestre no pueden tener clases al mismo tiempo.

Definimos la función auxiliar:

$$\rho(p_i, p_o, s_k) = n_{ki} n_{ko}$$

Por tanto: $\rho(p_i, p_o, s_k) = \begin{cases} 1 & \text{si el paralelo } i \text{ y el paralelo } o \text{ pertenecen al semestre } k \\ 0, & \text{caso contrario} \end{cases}$

$$\sum_{i=1}^{n_2-1} \sum_{o=i+1}^{n_2} \sum_{m=1}^{n_6} \sum_{k=1}^{n_4} \phi(p_i, h_m) \phi(p_o, h_m) \rho(p_i, p_o, s_k) = 0$$

[H9] La duración asignada para cada clase debe ser respetada.

$$N_k(c_j) = \sum_{i=1}^{n_6} k_{ij} = D(c_j); \forall j \in \{1 \dots \dots n_5\}$$

[H10] Los periodos asignados a una clase deben ser consecutivos

$$v(h_i) = \begin{cases} f_{i+1} - f_i - 1 & \text{si } f_{i+1} \text{ y } f_i \text{ pertenecen al mismo dia} \\ 0 & \text{caso contrario} \end{cases}$$

$$\sum_{i=1}^{n_6} \sum_{j=1}^{n_5} v(h_i) k_{ij} = 0$$

[H11] Las clases de los paralelos de la mañana deben ser dictadas en la mañana

Definimos la función:

$$\chi(h_i) = \begin{cases} 1 & \text{si la hora } f \text{ del periodo } h_i \text{ cumple con } 7 \leq f \leq 13 \\ 0 & \text{caso contrario} \end{cases}$$

Por tanto debe cumplirse que:

$$\sum_{i=1}^{n_2} \sum_{l=1}^{n_5} b_{ij} k_{lj} = \sum_{i=1}^{n_2} \sum_{l=1}^{n_5} \chi(h_l) b_{ij} k_{lj}$$

[H12] Las clases de los paralelos de la tarde deben ser dictadas en la tarde

$$\omega(h_i) = \begin{cases} 1 & \text{si la hora } f \text{ del periodo } h_i \text{ cumple con } 13 < f \leq 21 \\ 0 & \text{caso contrario} \end{cases}$$

Por tanto debe cumplirse que:

$$\sum_{i=1}^{n_2} \sum_{l=1}^{n_5} b_{ij} k_{lj} = \sum_{i=1}^{n_2} \sum_{l=1}^{n_5} \omega(h_l) b_{ij} k_{lj}$$

Aulas:

[H13] En la misma aula no se pueden dictar dos clases al mismo tiempo.

$$\tau(a_k, h_i) = \sum_{j=1}^{n_5} b_{ij} y_{jk}$$

$$\tau(a_k, h_i) = \begin{cases} 1 & \text{si el aula } k \text{ tiene clase en el periodo } i \\ 0 & \text{caso contrario} \end{cases}$$

Por tanto debe cumplirse que:

$$\tau(a_k, h_i) \leq 1 \quad \forall k, i$$

[H14] Una clase solo debe asignarse al tipo de aula indicada previamente.

$$\lambda(a_j) = \text{Tipo de aula del aula } a_j$$

$$\mu(c_i) = \text{Tipo de aula que necesita la clase } c_i$$

$$\delta(a_j, c_i) = \begin{cases} 0 & \text{si } \lambda(a_j) = \mu(c_i) \\ 1 & \text{si } \lambda(a_j) \neq \mu(c_i) \end{cases}$$

Por tanto debe cumplirse que:

$$\sum_{i=1}^{n_5} \sum_{j=1}^{n_3} \delta(a_j, c_i) = 0$$

2.3.5. RESTRICCIONES SUAVES

[S1] Preferencia profesor materia

$$\text{minimizar } f_1 = \sum_{i=1}^{n_7} \sum_{j=1}^{n_1} \sum_{k=1}^{n_2} e_{ij} l_{ki}$$

[S2] Compacidad de Horarios de Semestre

Definimos:

$$\beta(s_k, p_i) = \sum_{j=1}^{n_1} g_{ij} l_{kj}$$

$$\text{minimizar } f_2 = \sum_{k=1}^{n_4} \sum_{i=1}^{n_2} \sum_{n=1}^{n_6} \beta(s_k, p_i) \phi(p_h, h_n) v(h_n)$$

[S3] Materias concatenadas deberían ser calendarizadas al mismo tiempo

$$\text{minimizar } f_3 = \sum_{i=1}^{n_1-1} \sum_{j=i+1}^{n_1} \sum_{k=1}^{n_2} \sum_{m=1}^{n_6} z_{ij} g_{ki} \phi(p_i, h_o)$$

[S4] Compacidad de horario para los Alumnos

$$\text{minimizar } f_4 = \sum_{i=1}^{n_2} \sum_{o=1}^{n_6} \sum_{j=1}^{n_8} w_{ij} \phi(p_i, h_o) v(h_o)$$

[S5] Compacidad de horario para los Profesores

$$\text{minimizar } f_5 = \sum_{i=1}^{n_2} \sum_{o=1}^{n_6} \sum_{j=1}^{n_7} l_{ij} \phi(p_i, h_o) v(h_o)$$

2.4. CRITERIOS PARA ELEGIR EL ALGORITMO MÁS ADECUADO AL PROBLEMA

Como ya se dijo antes el problema de calendarización de horarios en el ámbito universitario es un problema de optimización combinatoria bastante restringido.

Para mostrar un ejemplo de la cantidad de posibilidades que se presentan al elegir una tupla que dé solución al problema se tomará como ejemplo los datos de la facultad de ingeniería de sistemas de la EPN.

Existen en el edificio de 15 aulas que se usan cotidianamente para las clases que reciben los alumnos, además existen 8 semestres dentro de la carrera, cada de los cuales tiene 7 materias dentro de su currículo en promedio, lo que nos da un total de 56 materias a las que los estudiantes atienden regularmente a clases. Cada una de estas materias cuenta al menos con 2 paralelos (uno en la mañana y uno en la tarde). Multiplicando estos dos números obtenemos un total de 112 paralelos activos dentro de un periodo de trabajo de la facultad.

Si además se toma en cuenta que cada paralelo tiene como promedio aproximado dos clases a la semana obtenemos un total de 224 clases a la semana para la facultad.

Dado que la facultad posee 15 aulas en las cuales es posible dictar clases, el número de combinaciones posibles para las clases que es posible dictar simultáneamente en cualquier periodo del horario es igual a:

$$C_{15}^{224} = \binom{224}{15} = \frac{224!}{(224-15)! 15!} = 8493852995709900000000,00$$

Aunque esta cantidad sea una aproximación muy inexacta, y se hayan cometido errores de cálculo con un aumento en el orden de los miles de millones, la cifra aun seguiría siendo muy alta como para intentar realizar una exploración completa de todas las posibilidades dentro del conjunto de datos.

Esta cantidad de operaciones necesarias para explorar por completo el total del conjunto de datos disponibles, nos obligaría a realizar búsquedas incompletas dentro de este conjunto de datos, de manera que sea posible encontrar una solución válida (u optima) explorando la menor cantidad de datos posible, y dentro de un tiempo aceptable.

Esto en definitiva es la esencia de todas las heurísticas, por lo que todo nos conduce a pensar que el problema sería un candidato ideal para el uso de estas. Existen una infinidad de heurísticas aplicables en estos días, todas con diferentes ventajas y debilidades; lo que las hace más o menos adecuadas para solucionar diferentes tipos de problemas.

Como ya se ha recalado antes el problema al que nos enfrentamos es un problema de optimización combinatoria bastante restringido.

La literatura existente menciona con más insistencia 5 tipos diferentes de heurísticas para solucionar problemas de optimización combinatoria, estos son:

1. Búsqueda local.
2. Búsqueda tabú.
3. Algoritmos Genéticos.
4. Swarm Intelligence.
5. Recocido simulado.

La búsqueda local ha probado ser una heurística versátil, y con buenos tiempos de convergencia para problemas con muchas restricciones; pero casi en ninguno de los estudios consultados se usa de manera independiente casi siempre se usa en combinación con otras heurísticas. Un problema serio que debe ser superado

cuando se usa la búsqueda local es la tendencia a quedar estancados en un óptimo local, que se toma como un óptimo global y se dejan de explorar las demás particiones del espacio de búsqueda que podrían contener un óptimo global.

La búsqueda tabú, está muy relacionada con la búsqueda local, tanto que se puede pensar en la búsqueda tabú como una búsqueda local combinada con estructuras en memoria de corto plazo. Estas estructuras se usan para evitar regresar hacia óptimos locales que ya se han evaluado. Un problema con la búsqueda local y las meta heurísticas basadas en esta, es que dado que son técnicas locales no exploran el espacio de solución de manera adecuada y pasan la mayor parte del tiempo explorando pequeños sectores del espacio de búsqueda y aunque convergen de manera rápida la convergencia puede darse hacia soluciones que están lejos de ser óptimas. Dentro de la búsqueda tabú, a pesar de la evidente ventaja que proveen los tabús esta situación debe remediarse haciendo uso de algoritmos de diversificación, por tanto la elección de estos, así como su implementación es una parte crítica de la puesta en marcha de las heurísticas basadas en búsqueda tabú.

Se ha probado que los algoritmos genéticos, cuyas soluciones son encontradas de manera totalmente estocástica tienden a no tener buenos tiempos de convergencia, cuando el probable espacio de solución es muy extenso. Los algoritmos genéticos han probado ser de mucha utilidad cuando no se tiene una visualización clara de cómo conseguir la solución del problema, y se poseen muy pocos elementos de juicio para elaborar una buena función de aptitud (objetivo). Usados en conjunto con redes neuronales han sido una poderosa herramienta para la solución de problemas de la vida real. No son de mucha utilidad para el manejo de restricciones duras, por lo que deben usarse acompañados de otro procedimiento que maneje este aspecto.

Recocido simulado, este método probabilístico fue el primero en atraer mucha atención sobre el campo de meta heurísticas, aplicadas hacia la solución de problemas de optimización combinatoria, de cierto modo se puede decir que los métodos de elección probabilística en los que se basa gran parte de las heurísticas de swarm intelligence son herederos de las formulaciones primeramente establecidas

por el recocido simulado. Entre las desventajas de esta meta heurística pionera tenemos largos tiempos de convergencia especialmente cuando se enfrenta a los grandes espacios de búsqueda relacionados con los problemas NP-completos, además como con otras meta heurísticas ya mencionadas es común tomar un óptimo local como uno global y quedar atrapados ahí.

La literatura menciona a las heurísticas de swarm intelligence como las más prometedoras dentro del campo de optimización combinatoria. Estas hacen uso extensivo de la stigmergia, es decir la colaboración espontánea e indirecta de varios agentes, donde a través de una acción se dejan rastros en el medio ambiente, que a su vez estimulan para que sean llevadas a cabo otras acciones, por el mismo o por un agente diferente, esta es una forma de auto-organización, que produce estructuras complejas y aparentemente inteligentes, sin necesidad alguna de planificación, control e incluso comunicación directa entre los agentes. En los últimos años se han realizado una cantidad considerable de investigaciones sobre las aplicaciones de estas para resolver problemas de optimización combinatoria surgidos de situaciones reales. Ofrecen buenos tiempos de convergencia y permiten manejar de manera bastante adecuada las restricciones suaves. Dentro de la multitud de heurísticas nacidas dentro del seno de swarm intelligence existe una que se destaca por la cantidad de investigaciones dedicadas: ACO (ANT COLONY OPTIMIZATION). Como ya se ha dicho de swarm intelligence, ACO también funciona muy bien para el manejo de las funciones objetivo, pero su tratamiento de las restricciones duras deja mucho que desear. Siendo como es un problema de optimización combinatoria restringido, la asignación de horarios presenta los dos tipos de restricciones. Una técnica complementaria para solucionar los problemas de optimización combinatoria restringidos es la Programación de Restricciones (CP), cuyas técnicas están especializadas para trabajar con las restricciones duras pero son ineficientes como método de optimización, especialmente si el espacio de búsqueda es grande.

Debido a todo lo antes mencionado, y después de una amplia consulta de la literatura disponible sobre el campo, para este trabajo se ha determinado que la combinación más prometedora, mezcla una heurística basada en ACO con la

programación de restricciones con la cual es posible realizar una búsqueda anticipada dentro del espacio de exploración y hace uso de los propagadores de restricciones lo que permite un buen manejo de las restricciones duras, y una reducción agresiva del espacio de búsqueda.

2.5. HERRAMIENTAS DE IMPLEMENTACIÓN

Las herramientas de implementación que se usarán dentro de este trabajo son las siguientes:

1. Documentos XML para guardar los datos de entrada.
2. C# como lenguaje para la implementación de los algoritmos desarrollados.
3. C# como lenguaje de implementación de la interfaz de usuario.

XML provee de flexibilidad y robustez al momento de definir estructuras, modelamiento de datos y verificación de los mismos. Además es portable (es decir no depende de la arquitectura del sistema operativo) y es posible exportar información de múltiples bases de datos hacia este formato.

Se usa C# para implementación de la interfaz de usuario debido a la rapidez con que es posible desarrollar aplicaciones sencillas que no requerían mucha personalización ni funcionamiento especializado.

La Figura 9 es el diagrama de componentes para el prototipo final:

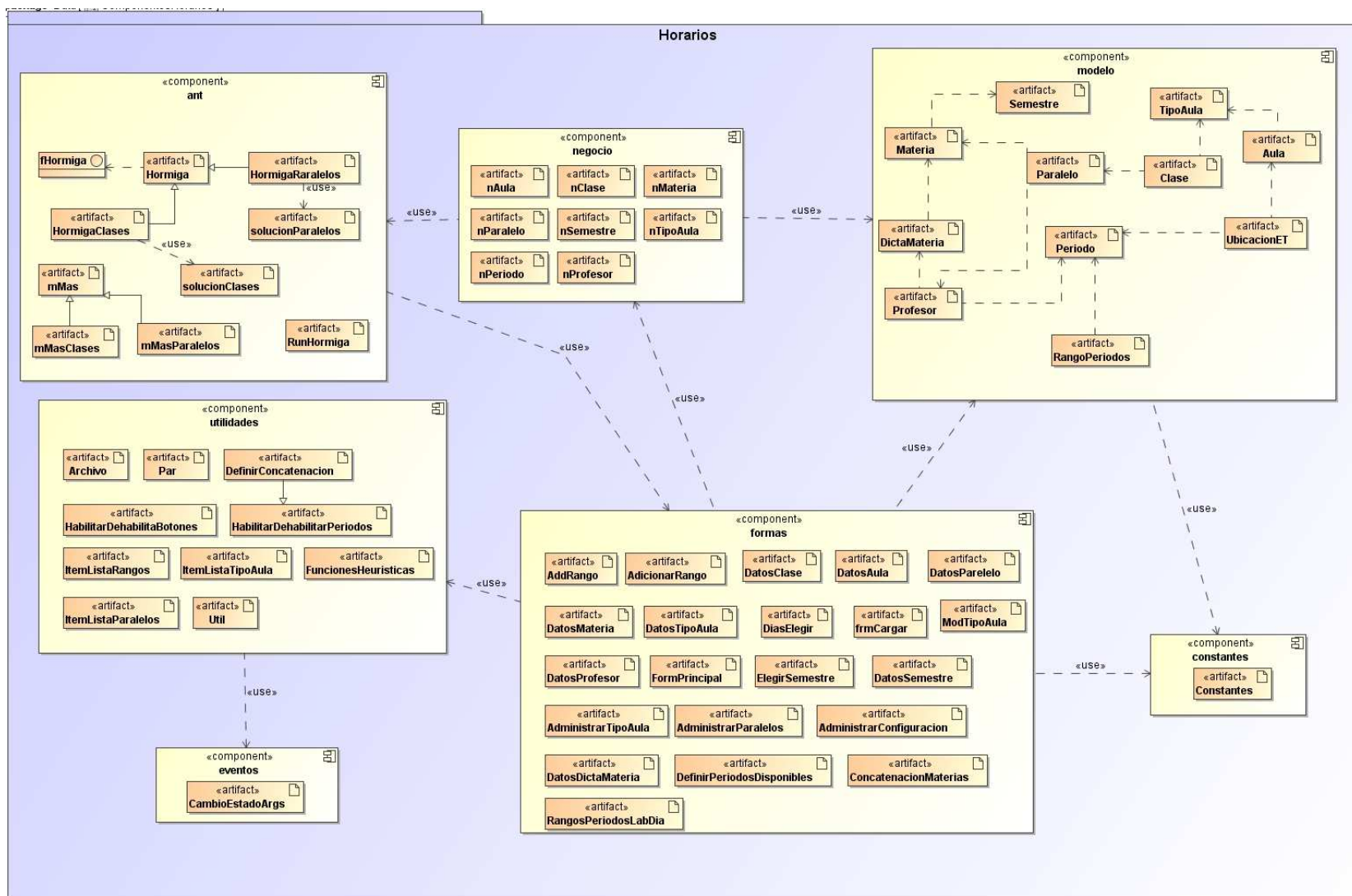


Figura 2.3 Diagrama de Componentes para el Prototipo

3. CAPÍTULO 3. CASO DE ESTUDIO

3.1. CARACTERIZACIÓN DEL CASO DE ESTUDIO

El caso de estudio aborda se basa en la Facultad de Ingeniería de Sistemas de la Escuela Politécnica Nacional. Históricamente los horarios se han venido realizando de manera manual, hace aproximadamente dos años se empezó a utilizar el SAE (Sistema de Administración Estudiantil) para permitir la matriculación en línea de los alumnos, el que consta de una interfaz administrativa que permite cargar los horarios en la aplicación, de manera que estén disponibles tanto para los estudiantes como para los profesores.

Esto abre una posibilidad para automatizar aun más el manejo de la información relacionada con los horarios al generarlos casi automáticamente y dejando que los encargados administrativos se encarguen de los detalles menores y finales de la puesta en marcha de la calendarización.

Actualmente dentro de la facultad existen profesores de tiempo parcial y tiempo completo; la mayor diferencia estriba en que los primeros únicamente tienen disponibilidad de tiempo fuera del horario de su trabajo principal y ejercen la actividad docente como complementaria, lo que incide directamente en la horas en que estos pueden dictar sus clases, mientras que para los profesores de tiempo completo esta es su actividad principal por lo cual su disponibilidad para dictar clases es mucho más amplia.

La facultad además cuenta con dos bloques horarios dentro de los cuales se dictan las clases de los paralelos en los diferentes semestres:

- Uno en la mañana es decir de 7 am a 13 pm.
- Y otro en la tarde y noche de 13 pm a 22 pm

Existen, así mismo, dos tipos de materias dentro del currículo de la facultad, las materias cuya aprobación es requisito indispensable en la obtención del título, que son las llamadas obligatorias y otras de las que se debe aprobar un número mínimo de créditos sin importar cuales estas sean de todas las disponibles, las llamadas optativas.

La mayoría de las materias se encuentra relacionada entre si es decir muchas de las materias de los semestres iniciales son prerrequisito de las materias en los semestres más adelante, es decir estas materias están concatenadas. Lo que

esto implica es que un estudiante no puede tomar al mismo tiempo clases en dos materias concatenadas, salvo la excepción de que tenga autorización de las autoridades.

Existen usualmente dos paralelos para la mayoría de materias que se dictan en la facultad, uno que tiene sus clases en la mañana y otro que tiene sus clases en el bloque de la tarde y noche. Sin embargo esto no quiere decir que exista siempre un máximo de dos paralelos por materia, ya que este número puede aumentar para acomodar las exigencias del número de estudiantes que solicitan inscripción. Tradicionalmente los diferentes paralelos de los semestres han gozado de compacidad dentro de los bloques horarios para las clases, es decir no han existido muchas horas libres o dispersión en los horarios sino que más bien los alumnos toman la clases de corrido hasta la finalización del bloque horario.

La facultad cuenta con aproximadamente 15 aulas normales para dictar clases diariamente a sus alumnos. Cuenta además con cuatro laboratorios disponibles para las clases prácticas de las diferentes materias, la suma de estos dos números debe tomarse muy en cuenta ya que nos dice cual es el máximo de clases que pueden realizarse simultáneamente.

Las aulas que más presión tienen en el uso de sus instalaciones son los cuatro laboratorios, ya que estos deben acomodar clases para prácticamente paralelos en todos los semestres, por este motivo son una pieza muy importante dentro de la elaboración de los horarios en la facultad.

Históricamente ha existido dentro de la EPN un alto índice de repetición entre los alumnos, y la Facultad de Sistemas no es la excepción, lo que hace que elaborar los horarios para esta sea todo un reto. Esto se debe a que una gran cantidad de estudiantes deben tomar clases en horarios extendidos, asistiendo tanto a paralelos de la mañana como de la tarde para completar el currículo académico en el menor tiempo posible. Esto además incrementa la probabilidad de que los estudiantes soliciten autorización para tomar materias concatenadas, de manera que también se incrementan las posibilidades de cruce dentro de los horarios de estos más allá de lo que podría considerarse como normal.

En primer término no se pensó en desarrollar una interfaz que sirva de enlace entre los algoritmos y las personas que los iban a usar, sino únicamente en implementar dos formatos en XML, el primero como medio de entrada de los datos provistos por los usuarios, y el segundo para mostrar los resultados hallados.

De esta manera se daría relevancia a un diseño más sofisticado de los modelos matemáticos y los algoritmos, pero también se pensó en omitir las GUI debido a la complejidad de implementar una aplicación gráfica usable que recepte la cantidad de datos requerida por los algoritmos, sin abrumar al usuario.

Al final se incluyó una interfaz para manejar el ingreso de datos, debido a que de lo contrario la aplicación restringía mucho el perfil del usuario objetivo, haciendo que solamente personas que tuvieran cierto conocimiento en el manejo de XML, además del necesario para la elaboración de horarios dentro ambientes universitarios, fuesen capaces de usar la herramienta con comodidad, lo que equivale a decir que esta no pasaba de ser un buen experimento.

Reduciendo los ambiciosos objetivos con respecto a la sofisticación de los algoritmos (el modelo matemático ya estaba terminado), fue posible desarrollar la interfaz de manera que contenga muchos aspectos de usabilidad, aunque como no se concibió de un inicio y por las restricciones de tiempo presentes en todo el desarrollo del proyecto, estos no estuvieron presentes desde el arranque, sino que se fueron incluyendo conforme avanzaba el proyecto.

Básicamente se tomaron en cuenta los siguientes aspectos de ingeniería de usabilidad dentro del desarrollo de las interfaces:

- Facilidad de aprender
- Eficiencia en el uso
- Rápida recuperación de errores
- Facilidad de recordarlas

Como se sabe la usabilidad es una característica que está relacionada con la medida de la calidad de los sistemas interactivos utilizados por usuarios

específicos en un contexto de trabajo determinado, para conseguir objetivos específicos en términos de utilidad, facilidad de uso, facilidad de aprendizaje y apreciación.

A pesar de que la creación de horarios implica una gran cantidad de ingreso de datos, se trató de incluir en las interfaces, efectividad y completitud para que el usuario alcance sus objetivos. Relacionados con estos conceptos está: la tasa de errores cometidos y la facilidad de recordar los usos de la aplicación.

Un aspecto que es de relevancia mencionar es además que se buscó desde un principio, fuentes de inspiración en la forma de otros desarrollos dentro del ámbito de automatización de horarios, como resultado de esto muchas de las interfaces están basadas en el paradigma que sigue “unitime.org”, desarrollo efectuado por el equipo de Tomáš Müller para la universidad de Purdue¹⁵.

Otra gran fuente de inspiración ha venido a través de la documentación generada por las conferencias del PATAT, lo que nos permitió tener una visión más clara de cómo se desarrolla el proceso de calendarización dentro del ambiente universitario en otros países.

El usuario modelo para el cual se ha basado el desarrollo tiene las siguientes características:

- Debe tener un nivel básico de conocimientos acerca de la elaboración de horarios.
- Debe poseer un nivel alto de conocimientos en el manejo de software dentro de ambiente Windows.

¹⁵ <http://www.unitime.org>; http://www.unitime.org/uct_demo.php

3.2. DEFINICION DE LOS TIPOS DE RESTRICCIONES EN EL CASO DE ESTUDIO

Se escogen las restricciones suaves y duras elaboradas para caso general que se adaptan de mejor manera al caso de estudio.

Las restricciones duras se han marcado con una H y las suaves con una S.

3.2.1. Restricciones Duras

H1: Se escoge como restricción dura debido a que ningún profesor que esté vinculado a la facultad puede quedar exento de dictar clases.

H2: Se escoge como restricción dura porque ningún paralelo que reciba clases, puede hacerlo sin un profesor. Se modifica un poco la restricción haciendo que cada paralelo este asignado a un profesor y solo uno.

Por tanto la ecuación es la que sigue:

$$\forall j \in \{1 \dots n_2\}, \sum_{i=1}^{n_7} l_{ij} = 1$$

H3: Es una restricción dura en todos los modelos debido a la imposibilidad física de estar en dos sitios al mismo tiempo.

H4: Se escoge como restricción dura, debido a que los profesores de tiempo parcial dedican los periodos en los cuales no dictan clases a sus actividades particulares, y les es imposible estar en los dos lugares al mismo tiempo.

H8: Se escoge como restricción dura, para evitar los cruces de horario entre materias del mismo semestre. La única excepción a esta regla serán dos paralelos de la misma materia.

H9: Se establece como una restricción dura, debido a que todas las clases tienen marcada una duración, definida mediante los objetivos curriculares de cada materia. Por este motivo esta duración debe respetarse en la creación de los horarios.

H10: Se establece como restricción dura debido a que las clases deben ser percibidas como unidades temporales por los estudiantes.

H11: Se escoge como restricción dura, porque si un paralelo ha sido catalogado como perteneciente al bloque horario de la mañana todas sus clases deben dictarse en ese bloque.

H12: Se escoge como restricción dura, porque si un paralelo ha sido catalogado como perteneciente al bloque horario de la tarde y noche (vespertino) todas sus clases deben dictarse en ese bloque.

H13: Se escoge como restricción dura debido a la incomodidad de mantener clases de materias diferentes en un aula al mismo tiempo.

H14: Se escoge debido a que las clases prácticas de muchas materias, pueden dictarse exclusivamente en los laboratorios equipados de manera adecuada para el efecto.

3.2.2. Restricciones Suaves

S1: Es deseable que los profesores dicten clases de materias que mas disfrutan y en las que están más capacitados, ya que esto propicia una mejor calidad de la enseñanza.

S2: Es deseable que los rangos de períodos en los cuales los estudiantes de cualquier semestre tienen clases no contengan horas libres y si las contienen sea en un número mínimo, ya que esto facilita el manejo del tiempo por parte de los estudiantes, los cuales pueden dedicar este tiempo a la realización de sus trabajos y tareas.

S3: Es deseable que las clases de paralelos de materias concatenadas estén calendarizados dentro de los mismos períodos, ya que se supone esto minimiza los cruces de horario.

S6: Decidimos combinar las restricciones duras **H5** y **H6** en una función objetivo, es decir convertirlas en una sola restricción suave, de manera que el problema altamente restringido, amplíe su espacio de solución, y por tanto las soluciones sean más factibles de encontrar.

Definimos:

$$f_6 = \begin{cases} a & \text{si } \sum_{i=1}^{n_2} \sum_{k=1}^{n_5} l_{ij} b_{ik} N_k(c_k) \leq H_t(t_j) \\ H_m(t_j) - \sum_{i=1}^{n_2} \sum_{k=1}^{n_5} l_{ij} b_{ik} N_k(c_k) & \text{si } H_t(t_j) \leq \sum_{i=1}^{n_2} \sum_{k=1}^{n_5} l_{ij} b_{ik} N_k(c_k) \leq H_m(t_j) \\ b & \text{si } \sum_{i=1}^{n_2} \sum_{k=1}^{n_5} l_{ij} b_{ik} N_k(c_k) \geq H_m(t_j) \end{cases}$$

Esta función tiene como objetivo que la carga horaria de los profesores sea siempre lo más cercana posible a la máxima carga horaria que cada profesor tiene designada.

Donde a y b son variables arbitrarias que se usan para darle mayor o menor peso a la función, según sea conveniente.

3.2.3. Restricciones de la interfaz de usuario

Debido a las restricciones de tiempo como ya se mencionó anteriormente y además debido a que esta es una aplicación experimental, esta no posee características a las que normalmente están disponibles dentro del software comercial.

Entre las limitantes dentro de esta primera versión del aplicativo se debe mencionar:

- No se provee de un esquema de seguridad, es decir la aplicación solo tiene identificado un usuario objetivo el que puede acceder libremente a toda la funcionalidad de la misma.
- No existe la posibilidad de deshacer o rehacer operaciones, es decir no se ha incluido la funcionalidad de los conocidos Ctrl+Z , Ctrl+Y , la que se encuentra disponible en la mayoría de aplicaciones de los principales Sistemas Operativos
- No se ha incluido atajos de teclado, es decir la mayoría de operaciones solo pueden efectuarse con el uso del ratón y no puede suplirse la falta de este usando únicamente el teclado.
- No se permite además la posibilidad de imprimir los resultados o los datos ingresados por los usuarios.
- No está disponible la personalización de la pantallas; los colores y el look and feel son los estándar de Windows, y no se provee manera de modificarlos.

3.3. MODELO MATEMATICO DEL CASO DE ESTUDIO

Las funciones que han sido sumadas para generar la función objetivo son las que ya han sido explicadas en la sección del modelo general, con la inclusión final de la función objetivo parcial formada con las restricciones duras H5 y H6. Los pesos que han sido incluidos servirán para dar mayor o menor relevancia a las funciones en el valor final de la función objetivo. Dado que se pretende minimizar la función objetivo, mientras menor valor tenga el peso asociado, mayor será la relevancia de la función parcial en el contexto de la función objetivo.

Las funciones de las restricciones también han sido tomadas de las realizadas para el modelo general.

Se ha decidido resolver el problema descomponiéndolo en dos fases. La primera se enfoca en la creación de paralelos con la información proporcionada de profesores y materias, las preferencias que estos tienen al dictar una u otra materia, la disponibilidad de horario que los profesores tienen y la carga horaria mínima y máxima de cada uno.

De esta manera es posible tratar con mayor eficacia la optimización de uso del tiempo de los profesores y además mejorar el índice de materias con alta preferencia que el profesor dicta.

La solución de este problema es un conjunto de paralelos, cuyas clases son las que se calendarizaran en la segunda parte de la solución.

En la segunda parte se toman los datos de los paralelos, cada uno de estos tiene asociado una materia, un profesor, y un conjunto de clases con una duración ya determinada, el objetivo en esta parte es asignar a cada una de clases de los diferentes paralelos, una ubicación espacio-temporal, respetando todas las restricciones duras y minimizando los valores de las funciones objetivo. Esta ubicación espacio-temporal no es más que la conjunción de un período con un aula en un par de la forma $u_i = (h_i, a_i)$.

Se definen dos modelos cada uno reflejando las restricciones duras y suaves que cada uno de los problemas toma en cuenta.

Así en la primera parte se toman en cuenta las restricciones suaves S1 (preferencia de los profesores sobre las materias) y S6 (maximizar la carga horaria de los profesores), combinadas, previa multiplicación por factores

heurísticos, para formar una función de optimización única. Las restricciones duras son: ningún profesor debe ser asignado a un paralelo con una materia para la cual no está capacitado para dictar clases, a pesar de ser una restricción dura se codifica dentro de la restricción suave S1, esto se ha hecho desde un inicio (dentro del modelo general) otorgándole un peso especial D a este caso, lo que hará que las soluciones que contengan este caso sean altamente indeseables. La siguiente restricción dura H1 (cada profesor debe ser asignado a por lo menos un paralelo) ha sido codificada dentro de la restricción suave S6. Esto es posible debido al hecho de que ningún profesor tiene designada como carga horaria máxima, un valor menor al número de créditos que han sido asignados para cada una de las materias. La última restricción dura H2 modificada (cada paralelo debe estar asignado a un profesor y solo uno), si debe ser verificada después de que se realiza cada asignación.

Por tanto el modelo para la primera parte es:

$$\text{minimizar } F_1 = Af_1 + Bf_6$$

Sujeta a:

H2:

$$\forall j \in \{1 \dots n_2\}, \sum_{i=1}^{n_7} l_{ij} = 1$$

La segunda parte tiene en su modelo las restricciones suaves S2 (Compacidad de los horarios del semestre) y S3 (Materias concatenadas deberían ser calendarizadas al mismo tiempo).

Las restricciones duras para esta etapa son:

H3 (Ningún profesor puede dar dos clases al mismo tiempo)

H4 (El profesor debe tener clases solo en los periodos en que está disponible)

H8 (Dos paralelos del mismo semestre no pueden tener clases al mismo tiempo)

H9 (La duración asignada para cada clase debe ser respetada)

H10 (Los periodos asignados a una clase deben ser consecutivos)

H11 (Las clases de los paralelos de la mañana deben ser dictadas en la mañana)

H12 (Las clases de los paralelos de la tarde deben ser dictadas en la tarde)

H13 (En la misma aula no se pueden dictar dos clases al mismo tiempo)

H14 (Una clase solo debe asignarse al tipo de aula indicada previamente)

Cada una de estas restricciones duras para esta etapa debe ser respetada, esto se hará en un proceso previo de tratamiento de los datos para eliminar del espacio de búsqueda los datos que no cumplan con las restricciones, y de ser necesario en tiempo de procesamiento del algoritmo principal de esta parte. La manera en que se descartarán los datos en el pre-procesamiento y se aplicaran las restricciones de la segunda parte se describe en la siguiente sección de este documento.

Por tanto el modelo para esta sección es el siguiente:

$$\text{minimizar } F_2 = Cf_2 + Df_3$$

Sujeta a:

H3:

$$\sum_{n=1}^{n_2-1} \sum_{m=n+1}^{n_2} \sum_{o=1}^{n_6} \phi(p_n, h_o) \phi(p_m, h_o) \theta(p_n, p_m) = 0$$

H4:

$$l_{ki} \phi(p_k, h_j) \leq q_{ij}, \forall i \in \{1 \dots n_7\}$$

H8:

$$\sum_{i=1}^{n_2-1} \sum_{o=i+1}^{n_2} \sum_{m=1}^{n_6} \sum_{k=1}^{n_4} \phi(p_i, h_m) \phi(p_o, h_m) \rho(p_i, p_o, s_k) = 0$$

H9:

$$N_k(c_j) = \sum_{i=1}^{n_6} k_{ij} = D(c_j); \forall j \in \{1 \dots n_5\}$$

H10:

$$\sum_{i=1}^{n_6} \sum_{j=1}^{n_5} v(h_i) k_{ij} = 0$$

H11:

$$\sum_{i=1}^{n_2} \sum_{l=1}^{n_5} b_{ij} k_{lj} = \sum_{i=1}^{n_2} \sum_{l=1}^{n_5} \chi(h_l) b_{ij} k_{lj}$$

H12:

$$\sum_{i=1}^{n_2} \sum_{l=1}^{n_5} b_{ij} k_{lj} = \sum_{i=1}^{n_2} \sum_{l=1}^{n_5} \omega(h_l) b_{ij} k_{lj}$$

H13:

$$\tau(a_k, h_i) \leq 1 \quad \forall k, i$$

H14:

$$\sum_{i=1}^{n_5} \sum_{j=1}^{n_3} \delta(a_j, c_i) = 0$$

3.4. ELECCION DEL ALGORITMO MÁS ADECUADO PARA EL CASO DE ESTUDIO

Como ya se dijo antes en la sección de criterios para elegir un algoritmo adecuado al problema en general, los algoritmos que se han visto como más adecuados son en realidad una mezcla de varios paradigmas.

Dentro del caso de estudio específicamente se ha escogido una heurística basada en ACO llamada MIN-MAX ant system, que establece límites en la cantidad de feromona que se permite colocar a las hormigas en una misma arista del grafo que estas recorren, lo cual evita que se caiga en el consabido problema del estancamiento en un óptimo local, al permitir la exploración de más aristas y que además muestra buenos tiempos de convergencia y un adecuado manejo de la restricciones suaves, las cuales se relacionan de manera fácil con la distancia entre nodos del grafo al ser una función directamente proporcional al valor de las funciones objetivo.

Se usa programación de restricciones para reducir el espacio de búsqueda donde sea posible de modo que las hormigas no recorran por completo grandes

conjuntos de datos sin necesidad, si no que únicamente exploren los subconjuntos que cumplen con las restricciones duras impuestas.

3.4.1. Configuración de los algoritmos

Los siguientes valores son los parámetros que guían a los algoritmos, estos son tanto valores de parametrización de número de hormigas e intentos a usarse dentro de las funciones principales tanto como de penalizaciones usadas dentro de las funciones objetivo, para las restricciones suaves:

Parámetro	Valor
NUMERO INTENTOS	3
NUMERO HORMIGAS CLASES	5
NUMERO HORMIGAS PARALELOS	5
INDICE DE EVAPORACIÓN	0,1
ALFA	2
BETA	2
INDICE MINIMO FEROMONA	1
PENALIZACION CARGA HORARIA INCORRECTA	50
PENALIZACION PARALELO NO ASIGNADO	50
PENALIZACION PROFESOR NO ASIGNADO	50
PENALIZACION HORAS NO CONTIGUAS	50

Tabla 3.1 Configuración de los Algoritmos¹⁶

Los valores mostrados arriba se han probado dentro del uso de la aplicación, siendo estos con los que mejores resultados se ha obtenido, por este motivo se los ha establecido como configuración por defecto dentro de la misma.

¹⁶ Los valores de los parámetros se ha obtenido de diferentes pruebas en el uso de la aplicación.

3.5. IMPLEMENTACION DEL ALGORITMO

El algoritmo se ha desarrollado para resolver el problema del caso de estudio consta de 4 partes:

1. **Toma de Datos y construcción de matrices de entrada:** En esta parte se toman los datos de aquellos introducidos por los usuarios en la interfaz construida para este propósito y se realizan algunos pasos de pre-procesamiento para construir las matrices de entrada que serán usadas.
2. **Resolución primera parte del modelo:** Se resuelve el primer problema de optimización planteado en la sección anterior.
3. **Pre-procesamiento previo a la segunda parte:** Con los datos obtenidos (paralelos y clases) se construye una matriz clase-ubicación-temporal cuyos datos respeten un subconjunto de las restricciones duras que son posibles de verificar en esta parte.
4. **Resolución segunda parte del modelo:** Resolución de la segunda parte, se minimiza la función objetivo de esta parte, respetando las restricciones duras que no han sido verificadas con anterioridad.

Como ya se dijo antes las clases que contienen el código del algoritmo de solución están implementadas en C# 3.5 el cual pertenece a Framework .Net 3.5.

Una de las ventajas que se encuentra en esta versión del framework .net y de C# es la inclusión de las funciones anónimas y tipos de datos dinámicos. Esto permite la programación en un estilo cercano a la programación funcional, es decir, permite la inclusión de código que tiene un estilo muy cercano al recomendado por este paradigma de programación, como es sabido este es usado extensamente en la programación de algoritmos de alto nivel y dentro de la investigación de ciencias de la computación.

Además esta implementación del lenguaje trae consigo una tecnología conocida como LINQ que permite realizar consultas similares a las que SQL ejecuta sobre

bases de datos relacionales, pero extendiéndolas para que las mismas se puedan efectuar sobre casi cualquier conjunto de datos que es posible definir dentro del lenguaje, en este caso C#. Esto nos brinda la oportunidad de aprovechar esta tecnología en la implementación de los propagadores de restricciones, que son los encargados de reducir los espacios de exploración.

Este proyecto también incluye una interfaz de usuario amigable, que permite usar el software de manera fácil y sencilla, la cual ha sido desarrollada aprovechando las facilidades que presta la herramienta Visual Studio 2008 de Microsoft.

3.5.1. Pseudo código del Algoritmo

Pseudocódigo de función principal

Parámetros de entrada:

T(lista de profesores)
 M(lista de Materias)
 P(lista de paralelos) (sin profesor asignado)
 C(listad de clases)
 A(lista de aulas)
 S(lista de semestres)
 H(lista de periodos disponibles)
 U(lista de rangos horarios)

Matrices de Entrada:

Paralelo-Materia: $G = P \times M$
 Paralelo-Clase: $B = P \times C$
 Profesor-Materia (preferencia): $E = T \times M$
 Periodo-Profesor: $F = H \times T$
 Semestre-Materia: $I = S \times M$
 Semestre-Aula (preferencia): $R = S \times A$
 Aula-Periodo (restricción): $O = A \times P$
 Materia-Materia: $Z = M \times M$

Parámetro de salida:

```
Solución (mejorSolucionEncontrada)
pmmas
inicializarProblemaMMAS(T,M,P,C,A,S,H,U,G,B,E,F,I,R,O,Z)
pmmas.InicializarFeromona()
mejor Conveniencia =999999
listaHormigas = inicializarHormigas()
mientras intento < Numero_Intentos
    intento = intento+1
Para Cada h ∈ listaHormigas
    hormigaRecorrer(h)
Fin Para Cada
evaporaFeromona()
Para Cada h ∈ listaHormigas
```

```

h.conveniencia = calcularConveniencia(h)
Si h.conveniencia < mejor Conveniencia entonces
    mejorSolucionEncontrada = h.Solucion
    mejorHormiga = h;
Fin Si
Fin Para Cada
mejorHormiga.depositarFeronoma()
pmmas.vigilarValoresMinMax()
devolver mejorSolucionEncontrada

```

Pseudocódigo de inicializar Hormigas

Parámetros de entrada:

NUMERO_HORMIGAS_DEFINIDO
 Problema Min Max Ant System pmmas

Parámetros salida:

listaHormigas
Para i=0 hasta i < NUMERO_HORMIGAS_DEFINIDO
 numeroAleatorio = generarnumeroAleatorio¹⁷
 listaHormigas[i] = nuevaHormiga(numeroAleatorio, pmmas)
Fin Para

Pseudocódigo de hormigaParaleloRecorrer

Parámetros de entrada:

Hormiga h
T(lista de profesores)
P(lista de paralelos) (sin profesor asignado)

Parámetros salida:

Ninguno
Para cada t ∈ T
 Si h.Solucion. ProfesorParalelos.contiene(t) == falso
 entonces
 h.Solucion. ProfesorParalelos.adicionar(t,
 listaVacia())
 listaProfesorParalelos = h.Solucion.ProfesorParalelos[t.Id]
 Para cada p ∈ P
 valorFeromona = pmmas.MatrizFeromona[t.Id, p.Id]^α
 valorHeuristica = pmmas.MatrizValoresHeuristica[t.Id, p.Id]^β
 indicadorCargaHoraria = calcularFuncionCargaHoraria(t)
 probabilidadEscogerParalelo = probabilidadEscogerParalelo + (
 valorFeromona * valorHeuristica * cargaHoraria)
 sumaFeromonaHeuristica = calcularSumaFeromonaHeuristica(t);
 cantidadComparar = numeroAleatorio * sumaFeromonaHeuristica ;
 Si probabilidadEscogerParalelo >= cantidadComparar **entonces**
 h.Solucion.ListaParesSolucion(nuevoPar(t.Id, p.Id))

¹⁷ Se usa una utilidad del framework .net

Fin Si
Fin Para Cada
Fin Para Cada

Pseudocódigo de depositar feromona

Para cada par \in ListaParesSolucion
 Ppmas.MatrizFeromona[par.Primerο, par.Segundo] =
 Ppmas.MatrizFeromona[par.Primerο, par.Segundo] +1
Fin Para Cada

Pseudocódigo de Vigilar Valores min max

Para cada $f_{ij} \in$ pmmas.MatrizFeromona
 Si $f_{ij} < \text{MinimoPermitidoFeromona}$ *entonces*
 $f_{ij} = \text{MinimoPermitidoFeromona}$
 Fin Si
 Si $f_{ij} > \text{MaximoPermitidoFeromona}$ *entonces*
 $f_{ij} = \text{MaximoPermitidoFeromona}$
 Fin Si
Fin Para Cada

Pseudocódigo de hormigaClaseRecorrer

Parámetros de entrada:

Hormiga hc
 C(lista de clases)
 H(lista de periodos)
 A(lista de aulas)
 D(lista días disponibles)
 T(lista de horas en el día)
 T'(Matriz de horas disponibles para la clase en el día)
 h'(Matriz de periodos por día escogidos para la clase)

Parámetros salida:

Para cada $c \in C$
 Para cada $d \in D$
 Para cada $t \in T$
 Si $T'_{dt} == \text{verdadero}$ *entonces*
 $h'_{ij} = (d, t)$
 Fin Si
 Fin Para Cada
 Fin Para Cada
 Para cada $t \in h'_i$
 Si $h'_{i+1} - h'_i > 1$ *entonces*
 Hc.Solucion.Deseabilidad=
 Hc.Solucion.Deseabilidad+valorCastigoNoContinuidad
 Fin Si
 hc.Solucion.ListaClasesAulas[i]=(c, aulaMayorPreferencia)
Fin Para Cada

3.5.2. PRUEBAS

Asegurar que todas las pruebas de la aplicación incluyendo ingreso, actualización de datos, búsquedas y procesos del negocio se cumplan satisfactoriamente.

3.5.2.1. Caso de Prueba: Definición de Datos

Clase:	Definición de Datos				
Requerimiento	Caso de Prueba		Operación	Esperado	Resultado
Ingresar Días a Elegir	Ingreso de días laborables	de	Almacenamiento de los datos en memoria	Datos Almacenados	Se realiza el almacenamiento
Ingresar Períodos Laborables en el día	Ingreso de períodos laborables	los	Almacenamiento de los datos en memoria	Datos Almacenados	Se realiza el almacenamiento

Tabla 3.2 Caso de Prueba: Definición de Datos

3.5.2.2. Caso de Prueba: Datos Semestres

Clase:	Datos Semestres				
Requerimiento	Caso de Prueba		Operación	Esperado	Resultado
Crear/Modificar Semestres	Ingreso de datos semestre	del	Almacenamiento de los datos en memoria	Datos Almacenados	Se realiza el almacenamiento

Tabla 3.3 Caso de Prueba: Datos Semestres

3.5.2.3. Caso de Prueba: Datos Materias

Clase:	Datos Materias			
Requerimiento	Caso de Prueba	Operación	Esperado	Resultado
Crear Materias	Ingreso de datos de las materias	Almacenamiento de los datos en memoria	Realizar la operación de almacenamiento	Se realiza el almacenamiento
Modificar Materias	Modificación de la Materia	Almacenamiento de los datos en memoria	Realizar la operación de modificación	Se realiza el almacenamiento

Tabla 3.4 Caso de Prueba: Datos Materias

3.5.2.4. Caso de Prueba: Datos Profesores

Clase:	Datos Profesores			
Requerimiento	Caso de Prueba	Operación	Esperado	Resultado
Crear Profesor	Creación del Profesor	Almacenamiento de los datos en memoria	Recibir los datos y realizar la operación de almacenamiento	Se reciben los datos y se realiza el almacenamiento
Modificar Profesor	Modificación del Profesor	Almacenamiento de los datos en memoria	Recibir los datos y realizar la operación de almacenamiento	Se reciben los datos y se realiza el almacenamiento
Definir Disponibilidad Profesor	Ingreso de información de disponibilidad del profesor	Almacenamiento de los datos en memoria	Recibir los datos y realizar la operación de almacenamiento	Se reciben los datos y se realiza el almacenamiento

Tabla 3.5 Caso de Prueba: Datos Profesores

3.5.2.5. Caso de Prueba: Datos Aulas

Clase:	Datos Aulas			
Requerimiento	Caso de Prueba	Operación	Esperado	Resultado
Crear Aula	Creación del Aula	Almacenamiento de los datos en memoria	Realizar la operación de almacenamiento	Se realiza el almacenamiento
Modificar Aula	Modificación del Aula	Almacenamiento de los datos en memoria	Realizar la operación de modificación	Se realiza la operación de modificación
Definir Disponibilidad Aula	Ingreso de información de disponibilidad del Aula	Almacenamiento de los datos en memoria	Recibir los datos y realizar la operación de almacenamiento	Se reciben los datos y se realiza el almacenamiento

Tabla 3.6 Caso de Prueba: Datos Aulas

3.5.2.6. Caso de Prueba: Archivo

Clase:	Archivo			
Requerimiento	Caso de Prueba	Operación	Esperado	Resultado
Guardar datos ingresados en un archivo XML	Guardar datos	Persistencia de los datos del modelo en el archivo XML	Realizar la operación de persistencia	Se realiza la operación satisfactoriamente.
Leer y cargar los datos de un archivo XML hacia memoria	Carga de Datos	Lectura de datos y carga en las estructuras de memoria	Realizar la operación de lectura y carga	Se realiza la operación satisfactoriamente.

Tabla 3.7 Caso de Prueba: Archivo

3.5.2.7. Caso de Prueba: Búsqueda Profesor

Clase:	Búsqueda Profesor			
Requerimiento	Caso de Prueba	Operación	Esperado	Resultado
Búsqueda de Profesor	Ingresando Criterio de Búsqueda	Realizar la búsqueda en base a la opción ingresada	Buscar todos los profesores que cumplan con el criterio de búsqueda	Se despliegan los resultados de la búsqueda.

Tabla 3.8 Caso de Prueba: Búsqueda Profesor

3.5.2.8. Caso de Prueba: Resultado Horario

Clase:	Resultado Horario			
Requerimiento	Caso de Prueba	Operación	Esperado	Resultado
Resultado de horarios	Listar los horarios de los recursos que fueron ingresados	Desplegar los horarios relacionados con los recursos definidos	Despliegue los horarios conformados para la revisión rápida de un resultado	Se despliegan los horarios.

Tabla 3.9 Caso de Prueba: Resultado Horario

3.5.3. CONCLUSIONES DE LAS PRUEBAS

- La realización de operaciones en memoria se efectuó de manera satisfactoria, debido a que el manejo de separación de aspectos permite tener un claro control sobre las operaciones que se efectúan sobre los datos.
- La modificación de los datos también se produjo de manera satisfactoria, aunque hubieron algunos errores antes de que estas pruebas fueran efectuadas en la etapa de desarrollo, que tenían que ver en su mayoría con la referencia de memoria de los objetos compartidos entre diferentes interfaces.
- El manejo de datos a través del archivo XML también se produjo de manera correcta, se puede concluir que la manipulación de datos a través de este medio a pesar que se realiza de manera casi manual es poco propensa a producir errores de consideración, esto puede deberse al absoluto control que el programador tiene sobre los datos.
- La inclusión de búsquedas de datos se produjo luego de una prueba informal con un usuario, el cual sugirió que esta característica podría ser muy útil debido a la cantidad de información que la aplicación contiene.
- Aunque los horarios se generan de manera correcta en la mayor parte de casos, existen ocasiones en que el ingreso de datos no coherentes sobre las restricciones conduce a planteamientos del problemas irresolubles, debido a esta situación se incluye funcionalidad que comprueba la coherencia de esta información antes de intentar generar soluciones al problema.

4. CAPÍTULO 4. CONCLUSIONES Y RECOMENDACIONES

4.1. CONCLUSIONES

- De acuerdo a nuestra experiencia la parte esencial en toda aplicación de optimización es en primer lugar tener una visión clara de los objetivos a alcanzar, es decir tener muy bien identificadas las variables y de qué manera estas van a ser optimizadas.
- El modelo, que abarca las variables involucradas dentro el problema, así como también las relaciones que rigen a estas, debe acercarse con la mayor exactitud posible a la realidad que se está tratando de abstraer con este. De esto depende el éxito de las aplicaciones que puedan basarse en el modelo.
- Las restricciones duras sirven para delimitar lo que se acepta como un resultado válido, por tanto es importante tratar de elegir la cantidad justa de estas; demasiadas y el problema será extremadamente difícil de solucionar; muy pocas y las soluciones obtenidas tal vez no sean de utilidad.
- Al igual que las restricciones duras, las restricciones suaves son muy importantes a la hora de definir el problema, estas (las restricciones suaves) ayudan a puntualizar que es aceptado como una solución deseable pero sin ser excluyentes como en el caso de las duras. Su número como sucede con las restricciones duras, debe estar equilibrado dentro del problema; muy pocas y parecerá que no tenemos suficientes funciones objetivo con las que trabajar; demasiadas y quizá tome mucho tiempo de procesamiento para que el problema sea resuelto.
- La única manera de obtener los valores de los parámetros de configuración es a través del antiguo sistema de prueba y error.

- Como se mencionó anteriormente se utilizaron archivos XML para la persistencia de la información. Esta elección se hizo debido a la portabilidad que este formato proporciona, y la posibilidad de migrar información desde diferentes orígenes. Pero la carga de mantener cada una de las operaciones CRUD (Create, Retrieve, Update, Delete), recae en la programación, este trabajo podría haberse visto facilitado mediante el uso de una base de datos.
- El paso de un modelo teórico hacia una implementación práctica, debe realizarse con cuidado, tratando de reflejar lo que se diseñó en papel de manera fidedigna, pero, y aunque suene contradictorio, debe tenerse la mente abierta a realizar cambios de último minuto que incorporen mejoras o permitan la correcta solución del problema.
- El intento de reducir el número de interfaces y la cantidad de información que el usuario debe ingresar para resolver el problema, no fue exitoso, debido a la cantidad de información indispensable que los algoritmos y el modelo requieren.
- Debido a las características de la aplicación, el tipo de usuario que se sentirá más cómodo usándola es una persona acostumbrada al uso cotidiano de aplicaciones de oficina y con un conocimiento intermedio de elaboración de horarios en el nivel universitario.
- A pesar de todo el trabajo que requirió desarrollar esta aplicación, está lejos del nivel de usabilidad y confiabilidad que normalmente requiere una aplicación para su lanzamiento comercial.
- La aplicación hace uso de múltiples validaciones a través de varias pantallas, lo que para algunos usuarios resulta molesto, pero estas son indispensables para obtener resultados coherentes.

- Como ya se dijo antes la aplicación, requiere del ingreso de mucha información, debido a este motivo se observó en las pruebas que los usuarios que trabajan con esta por primera vez requieren de una guía base, para poder efectuar las tareas con eficacia.
- Debido a la retroalimentación obtenida mediante las encuestas realizadas a los usuarios que probaron la aplicación, se determinó que ciertos mensajes de ayuda y errores eran confusos, por lo que requerían de revisión y mejora.
- Las nuevas características de programación funcional que están presentes dentro de la versión de C# usada probaron ser de mucha utilidad al momento de codificar los algoritmos. También fue de mucha utilidad la característica de consultas de colecciones de objetos con sintaxis SQL, conocida como LINQ, que así mismo está disponible en esta versión.

4.2. RECOMENDACIONES

- La realización de un modelo básico exitoso requiere un conocimiento de intermedio hacia arriba sobre programación lineal, así como también de algebra lineal y espacios vectoriales, por tanto si estos conocimientos adquiridos al principio de la carrera se han olvidado, es muy prudente realizar un exhaustiva revisión de estos temas.
- Sería recomendable que en versiones posteriores de la aplicación, se use una base de datos, debido a la mayor facilidad de manejo de la información y la mejora en rendimiento que podría obtenerse.

- De nuestra experiencia dentro del desarrollo de la aplicación, es muy recomendable tener separación de aspectos (separation of concerns) dentro de la implementación. Hemos observado que esto reduce significativamente la redundancia de código y permite dar mantenimiento a la aplicación con mayor facilidad.
- Podría ser interesante usar otras heurísticas, dentro de la implementación de la aplicación para poseer datos experimentales que corroboren la elección que se realizó a nivel teórico en este trabajo.
- Es recomendable usar lenguajes con características de programación funcional en la implementación de los algoritmos, los cuales debido a su sintaxis proveen características que facilitan el trabajo de manera significativa.
- Sería recomendable tener archivos de recursos donde se centralicen los mensajes mostrados por la aplicación ya que esto facilita el mantenimiento de los mismos.
- A futuro sería recomendable desarrollar motores de solución de problemas combinatorio que se puedan usar intercambiamente dentro de la aplicación, siendo esta operación lo más transparente posible para esta, Esto sería de mucha utilidad ya que la práctica muestra que algunos algoritmos se comportan mejor que otros, dependiendo de las peculiaridades de cada problema.
- Para que la aplicación pueda intercambiar información con otro software de manera segura, es bastante recomendable establecer una validación de los datos mediante esquemas o documentos de definición de datos (DTD), de modo que se muestren de manera explícita todos los datos, sus tipos, y las restricciones y validaciones que se efectúan sobre estos.

BIBLIOGRAFÍA

Libros:

- DORIGO Marco, CORNE David, GLOVER Fred. New ideas in optimization. McGraw-Hill. Londo.1999
- DORIGO, Marco. Ant Colony Optimization. Cambridge Massachusett. 2004.

Tesis:

- Cruz, H., & Viteri, V. (2007). Optimización de Problemas Combinatorios y Multiobjetivo Utilizando el Método Colonia de Hormigas. Quito, Abril 2007.

Direcciones Electrónicas:

- Chávez, O., & De los Santos, G. (2005). Búsqueda Tabú aplicada a un Problema NP-completo
<http://usuarios.multimania.es/chavezbosquez/arch/docs/Conferencia2005.pdf>
- Fidanova, S., & Durchova, M. (2005). Ant Algorithm for Grid Scheduling Problem
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.95.2775&rep=rep1&type=pdf>
- Mayer, A., & Nothgger, C. (2007). Solving the Post Enrolment Course Timetabling Problem by Ant Colony Optimization.
http://w1.cirrelt.ca/~patat2008/PATAT_7_PROCEEDINGS/Papers/Chwatal-WA1b.pdf
- Ritchie, G. (2003). Static Multi-processor Scheduling with Ant Colony Optimization & Local Search
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.6.2601&rep=rep1&type=pdf>
- Rossi-Doria, O. (2004). A comparison of the performance of different Metaheuristics on the timetabling problem
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.83.5836&rep=rep1&type=pdf>
- Rudova, H., & Murray, K. (2004). University Course Timetabling with Soft Constraints
<http://www.unitime.org/papers/patat02.pdf>

- Seok, K., & Woo, Z. (30 de 09 de 2004). A new meta-heuristic algorithm for Continuous engineering
<http://ftp.ltu.edu.tw/~mhc/Special%20Topic/references/papers/HS/A%20new%20metaheuristic%20algorithm%20for%20continuous%20engineering%20Optimization%20harmony%20search%20theory%20and%20practice.pdf>
- Socha, K. (31 de 03 de 2003). MAX-MIN Ant System for International Timetabling Competition
<http://www.idsia.ch/Files/ttcomp2002/socha.pdf>
- Socha, K., Sampels, M., & Manfrin, M. (2003). Ant Algorithms for the University Course Timetabling Problem with Regard to the State-of-the-Art.
<http://www.iihe.ac.be/internal-report/2003/stc-03-06.pdf>
- Stutzle, T. (2003). Iterated Local Search — Variable Neighborhood Search.
<http://www.sls-book.net/Slides/sls-ils+vns.pdf>
- Stützle, T., & Hoos, H. (Mayo de 1994). MAX-MIN Ant System and Local Search
<http://staff.washington.edu/paymana/swarm/stutzle97-icec.pdf>
- Vittorio, M., & Gambardella, M. (2004). 5. Ant Colony Optimization.
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.10.7560&rep=rep1&type=pdf>
- Lach, M. (February 27, 2008)Curriculum Based Course Timetabling Optimal Solutions to the Udine Benchmark Instances
www.math.tuberlin.de/~luebbeck/papers/udine.pdf
- Daskalaki, T. & Birbas, E. (2004)An integer programming formulation for a case studying university timetabling
<http://www.ceet.niu.edu/faculty/ghrayeb/IENG576s04/papers/IP/An%20integer%20programming%20formulation%20for%20a%20case%20study.pdf>
- Miranda, P. & Morales, A. (2008) Un modelo de programación entera para la calendarización y generación de horarios para la escuela de educación ejecutiva de la universidad de chile
<http://bibliotecas.uchile.cl/documentos/20081012-1626569236.pdf>
- Geraldo R. & Nogueira L. An Integer Programming Model for the School Timetabling Problem
<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.69.751>
- Minhaz. & Fahim. A multi-phase approach to university course timetabling
<http://www.uleth.ca/dspace/bitstream/10133/633/1/zibran,%20minhaz.pdf>

- Koyuncu, B. & Secir M. Student time table by using graph coloring algorithm
http://www.emo.org.tr/ekler/76e76856c7fea3b_ek.pdf
- Socha, K. & Birattari, M. International Timetabling Competition A Hybrid Approach
<http://www.intellektik.informatik.tudarmstadt.de/~machud/files/AIDA-03-04.pdf>
- Clark, M. & Henz, M. QuikFix A Repair-based Timetable Solver
<http://www.comp.nus.edu.sg/~henz/publications/ps/PATAT2008.pdf>
- Bello, G. & Rangel, M. An approach for the Class/Teacher Timetabling Problem using Graph Coloring
<https://symposia.cirrelt.ca/system/documents/.../PROGRAM-12aout-WEB.pdf>
- Chiarandini, M. & Fawcett C. A Modular Multiphase Heuristic Solver for Post Enrolment Course Timetabling
http://www.cs.ubc.ca/~hutter/earg/presentations/March12_ITCResults.pdf
- Burke, E. & Elliman D. & Weare R. A University Timetabling System based on Graph Colouring and Constraint Manipulation
<http://www.asap.cs.nott.ac.uk/publications/ps/jrce.ps.gz>
- Silva, J. & Petrovic, S. & Burke, E. An Introduction to Multiobjective Metaheuristics for Scheduling and Timetabling
http://www.asap.cs.nott.ac.uk/publications/pdf/JDLS_LNEMS2004.PDF
- Massoodian, S. & Esteki, A. (2008) A Hybrid Genetic Algorithm for Curriculum Based Course Timetabling
http://w1.cirrelt.ca/%7Epatat2008/PATAT_7_PROCEEDINGS/Papers/Massoodian-TC1b.pdf

ANEXOS

ANEXO 1

MANUAL DE USUARIO DEL SISTEMA

Manual de Usuario

Método para la resolución de problemas de Calendarización utilizando el enfoque “Optimización de Colonia de Hormigas”

Tabla de Contenido

1. PANTALLA PRINCIPAL	2
2. DEFINICIÓN DE DATOS.....	2
2.1. Días a Elegir.....	2
2.2. Rango de Períodos Laborables.....	3
2.3. Definir Períodos.....	4
2.4. Administrar tipos de Aulas.....	5
2.5. Añadir Nuevo Tipo de Aula.....	6
2.6. Configuraciones	6
3. ADMINISTRAR SEMESTRES, MATERIAS, PARALELOS Y CLASES	7
3.1. Datos de los Semestres	7
3.2. Datos de la Materia	8
3.3. Administrar Paralelos	9
3.4. Añadir Nuevo Paralelo.....	9
3.5. Datos de la Clase	10
3.6. Elegir Semestres para concatenación de Materias	11
4. ADMINISTRACIÓN DE PROFESORES.....	13
4.1. Datos del Profesor.....	13
4.2. Definir Períodos disponibles del Profesor.....	14
4.3. Preferencia de las Materias que puede dictar el profesor	15
5. ADMINISTRACIÓN DE AULAS	16
5.1. Datos del Aula	16
5.2. Definir Períodos Disponibles de las Aulas.....	17
5.3. Definir Preferencia Aulas.....	18
6. PROCESAMIENTO DEL ALGORITMO	19
6.1. Abrir, Guardar los datos	19
6.2. Proceso para obtener resultados	20

5. PANTALLA PRINCIPAL

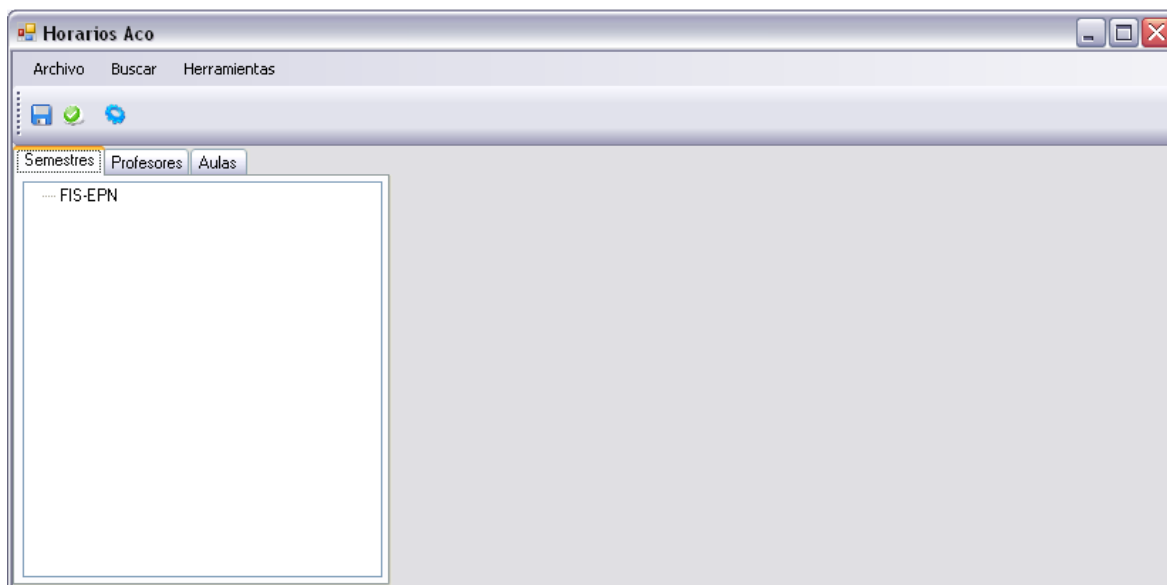


FIGURA 4 PANTALLA PRINCIPAL

En la figura 1 se muestra la Pantalla Inicial: Calendarización de Horarios en la cual se ingresarán los semestres, profesores, aulas, clases, paralelos y las diferentes definiciones de datos mostradas a continuación.

6. DEFINICIÓN DE DATOS

6.1. Días a Elegir

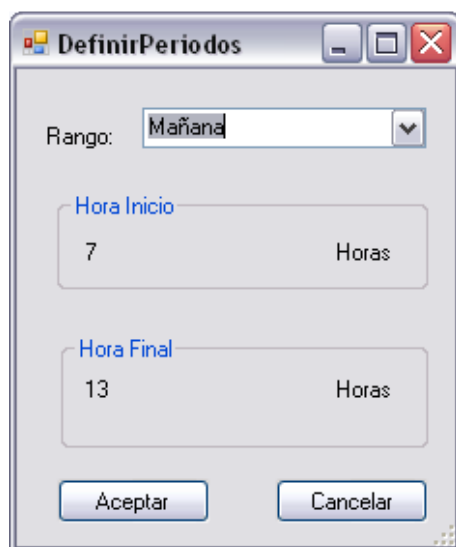


FIGURA 5 DÍAS A ELEGIR

resumen de los rangos añadidos dentro de la definición del problema. Se debe ir a la opción herramientas y luego a “Definir Períodos Laborables en el día”.

Para añadir un nuevo rango horario se debe dar clic en el botón Añadir Nuevo Rango el cual se muestra en la figura 4. Para eliminar un rango horario se debe seleccionar un rango horario ingresado y luego clic en el botón Eliminar Rango. Los botones Aceptar y Cancelar nos permiten aceptar o cancelar los rangos horarios ingresados.

6.3. Definir Períodos

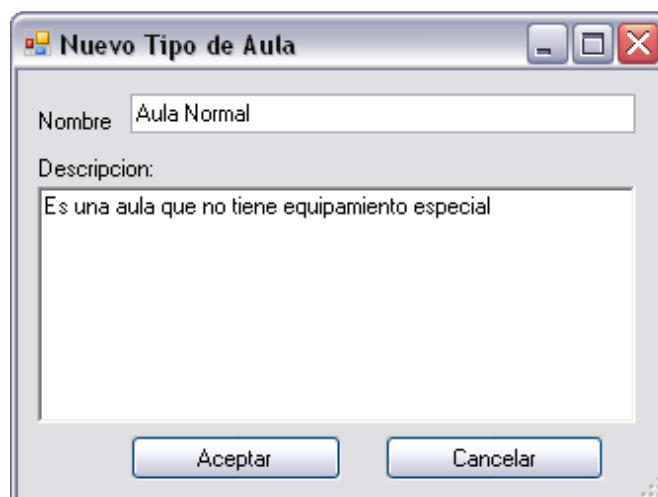


The image shows a Windows-style dialog box titled "DefinirPeriodos". At the top, there is a "Rango:" label followed by a dropdown menu currently displaying "Mañana". Below this, there are two input fields. The first is labeled "Hora Inicio" and contains the number "7", with the word "Horas" to its right. The second is labeled "Hora Final" and contains the number "13", also with "Horas" to its right. At the bottom of the dialog, there are two buttons: "Aceptar" on the left and "Cancelar" on the right.

FIGURA 7 DEFINIR PERÍODOS

En la figura 4 se muestra el ingreso para cada uno de los rangos horarios, en la opción Rango se puede escoger tres tipos de rango: Mañana (7 a 13 horas), Tarde (13 a 18 horas) y Noche (18 a 22), cada que se escoge un rango se muestra por default una hora de Inicio y una hora de Fin.

6.5. Añadir Nuevo Tipo de Aula

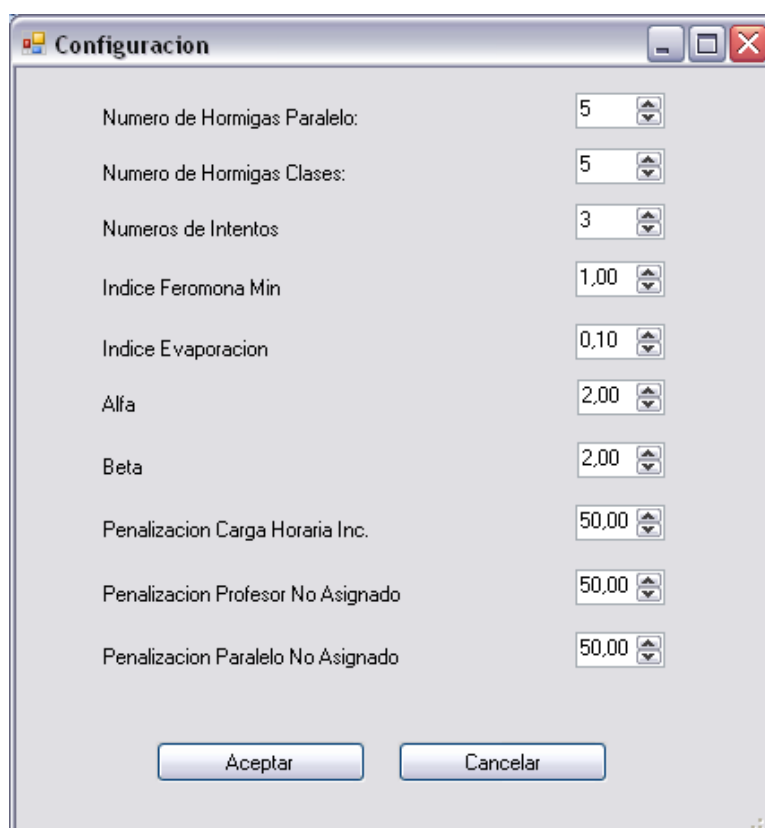


The dialog box titled "Nuevo Tipo de Aula" has a title bar with standard window controls. It contains a text input field for "Nombre" with the value "Aula Normal". Below it is a text area for "Descripcion" containing the text "Es una aula que no tiene equipamiento especial". At the bottom, there are two buttons: "Aceptar" and "Cancelar".

FIGURA 9 AÑADIR NUEVO TIPO DE AULA

En la figura 6 se puede ingresar los datos de cada tipo de aula, se ingresa el nombre del tipo de aula y una breve descripción acerca de cada tipo de aula. Clic en Aceptar.

6.6. Configuraciones



The "Configuracion" dialog box displays a list of parameters, each with a spin button to its right. The parameters and their values are: "Numero de Hormigas Paralelo" (5), "Numero de Hormigas Clases" (5), "Numeros de Intentos" (3), "Indice Feromona Min" (1,00), "Indice Evaporacion" (0,10), "Alfa" (2,00), "Beta" (2,00), "Penalizacion Carga Horaria Inc." (50,00), "Penalizacion Profesor No Asignado" (50,00), and "Penalizacion Paralelo No Asignado" (50,00). At the bottom, there are "Aceptar" and "Cancelar" buttons.

FIGURA 10 CONFIGURACIONES

La pantalla de la figura 7 muestra parámetros con valores por default los cuales serán utilizados por el algoritmo para la obtención de los resultados. Se debe ir a la opción herramientas y luego a “Configuraciones”.

7. ADMINISTRAR SEMESTRES, MATERIAS, PARALELOS Y CLASES

Definir dentro de la enunciación del problema los semestres, materias que pertenecen a cada uno de estos, los paralelos que a su vez pertenecen a las materias y las clases que están vinculadas a cada paralelo.

El ingreso de estos datos debe hacerse en un orden determinado siguiendo esta secuencia:

7.1. Datos de los Semestres

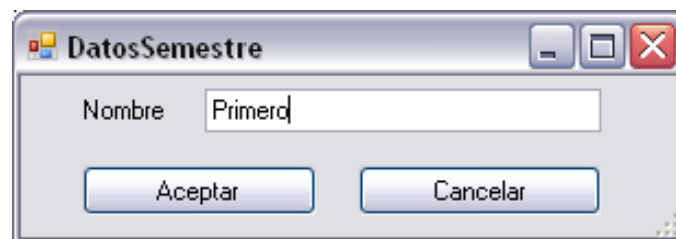
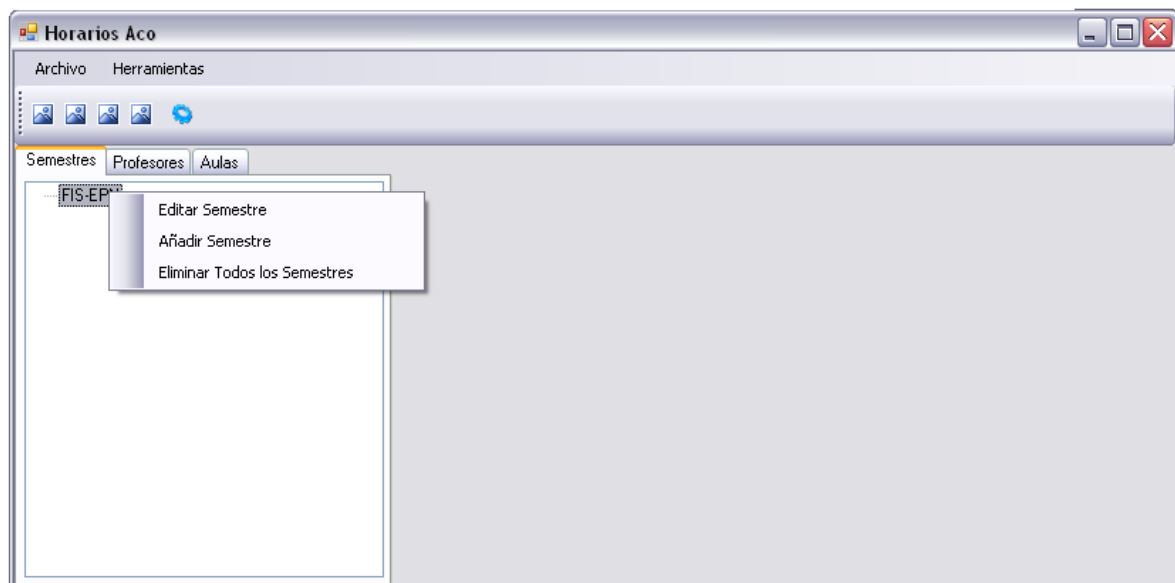


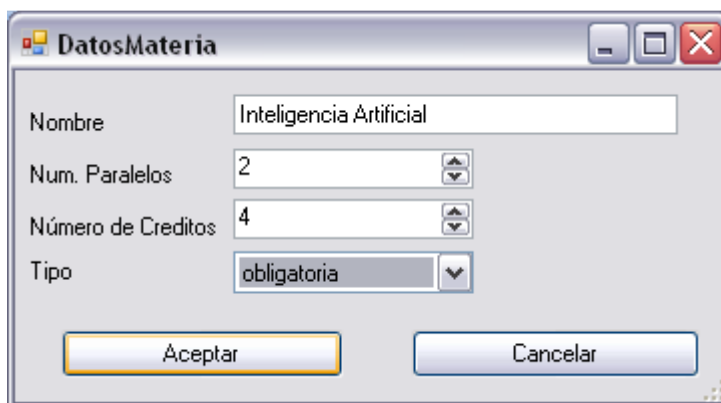
FIGURA 11 DATOS DEL SEMESTRE

Primero se ingresan los datos de los semestres como indica la figura 8, en la pantalla principal se debe ir a la opción semestres dar clic derecho en FIS-EPN “Añadir Semestre”. Damos clic en la opción Aceptar.



Además tenemos la opción Editar Semestre la cual nos permite modificar los datos ingresados del semestre, se debe dar clic derecho en el semestre y elegir la opción “Editar Semestre”.

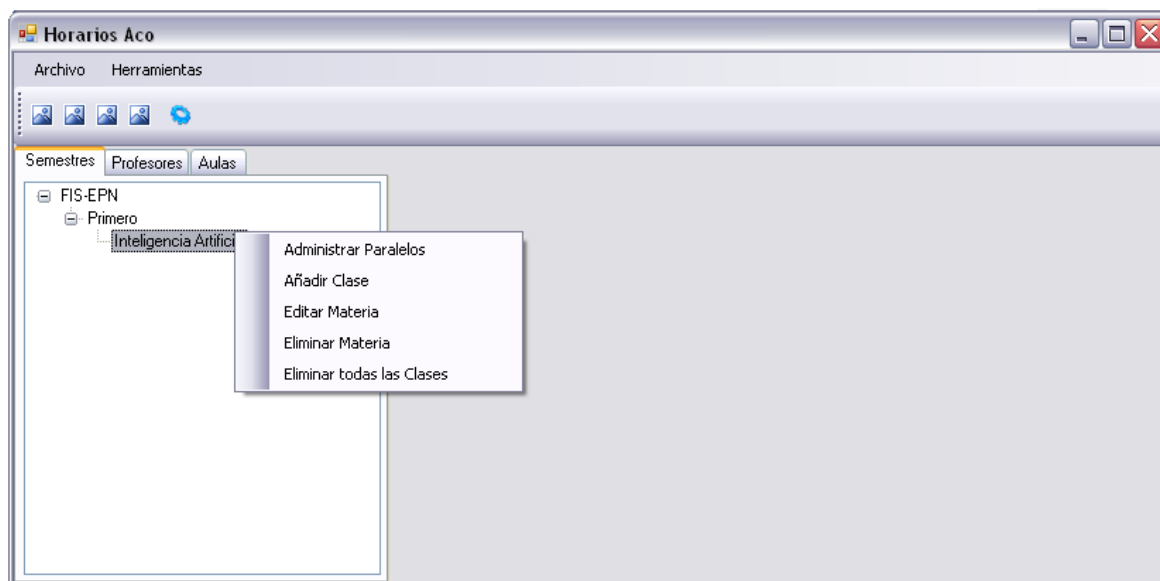
7.2. Datos de la Materia



The image shows a dialog box titled "DatosMateria". It has four input fields: "Nombre" with the text "Inteligencia Artificial", "Num. Paralelos" with the value "2", "Número de Creditos" with the value "4", and "Tipo" with the value "obligatoria". At the bottom, there are two buttons: "Aceptar" and "Cancelar".

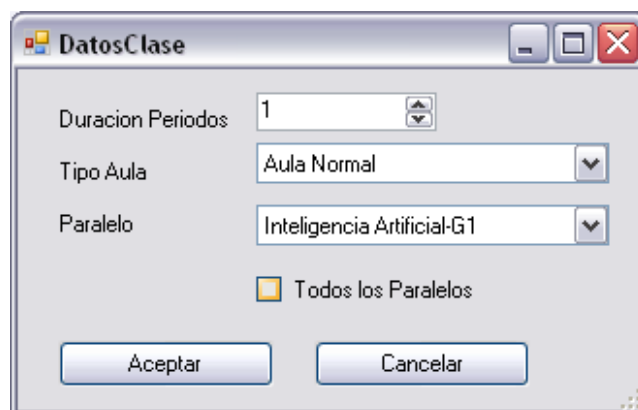
FIGURA 12 DATOS DE LA MATERIA

Luego para cada uno de los semestres debemos ingresar las materias que están asociadas, con los datos que vemos en la siguiente interfaz mostrada en la figura 9. Se ingresa el nombre de la materia, el número de paralelos, número de créditos y un tipo de materia el cual ya se encuentra establecido por default (obligatoria y opcional). Se debe dar clic derecho en cada semestre y elegir la opción “Añadir Materia”. También se puede Editar Materia la cual nos permite modificar los datos que ingresamos anteriormente



La interfaz de la figura 11 muestra cómo definir un paralelo, se debe elegir el rango horario en que sus clases se llevaran a cabo, en este caso aparecerán los rango horarios (Mañana y Tarde) que se definieron anteriormente en la definición de los datos. El Grupo aparecerá por default de acuerdo al número de paralelos que se definieron el momento de ingresar una materia.

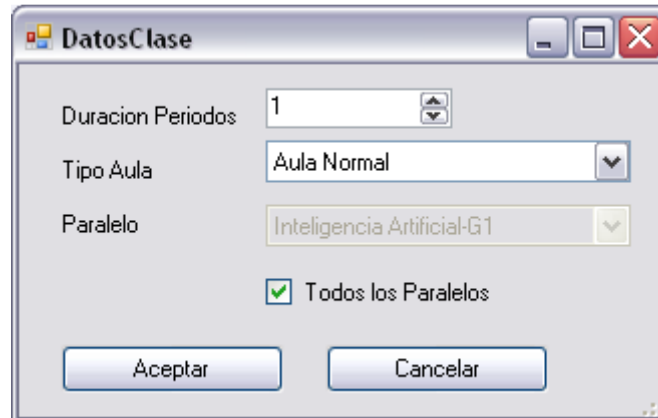
7.5. Datos de la Clase



The screenshot shows a dialog box titled "DatosClase" with the following fields and controls:

- Duracion Periodos:** A numeric input field containing the value "1".
- Tipo Aula:** A dropdown menu with "Aula Normal" selected.
- Paralelo:** A dropdown menu with "Inteligencia Artificial-G1" selected.
- Checkbox:** A checkbox labeled "Todos los Paralelos" which is currently unchecked.
- Buttons:** "Aceptar" and "Cancelar" buttons at the bottom.

FIGURA 15 DATOS DE LA CLASE (UN ÚNICO PARALELO)



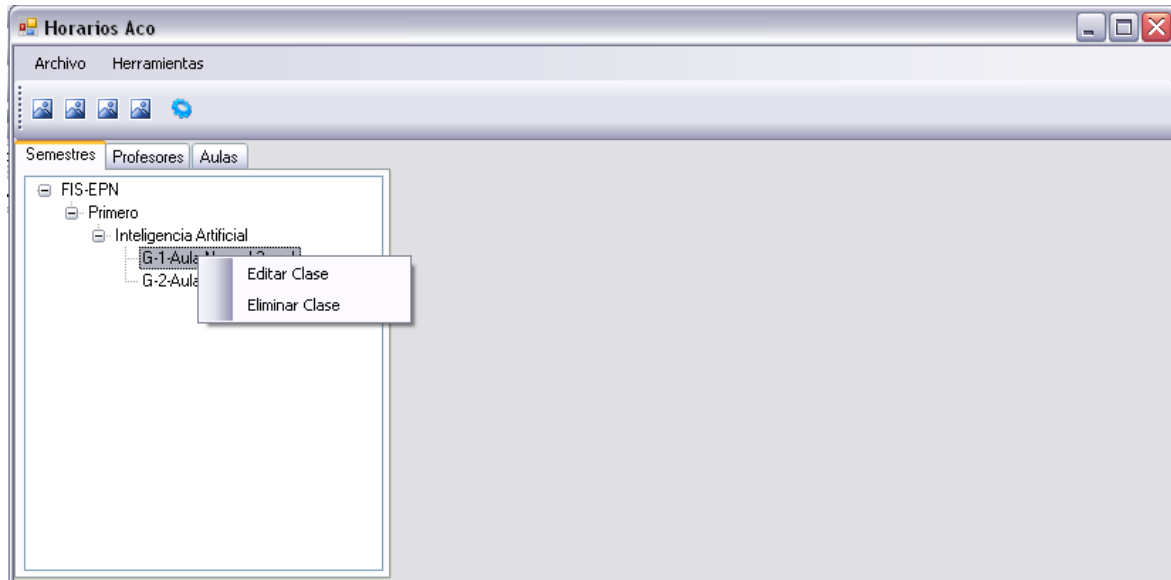
The screenshot shows the same "DatosClase" dialog box as in Figure 15, but with the following differences:

- Checkbox:** The checkbox labeled "Todos los Paralelos" is now checked.
- Buttons:** The "Aceptar" and "Cancelar" buttons remain at the bottom.

FIGURA 16 DATOS DE LA CLASE (TODOS LOS PARELELOS)

La pantallas 12 Y 13 se utilizan para ingresar los datos de las clases que constan dentro del problema. Para ingresar las clases se debe dar clic derecho en una materia ingresada y elegir la opción "Añadir Clase" como se muestra anteriormente. Se ingresa la duración de períodos esto depende de cuánto durará cada clase, se debe elegir un tipo de aula el cual se ingreso en la parte de definición de datos, la opción Paralelo tiene dos opciones: la figura 12 nos permite

elegir solo un paralelo registrado de una materia, mientras que en la figura 13 nos permite añadir automáticamente todos los paralelos registrados de una materia. También se puede Editar Clase que permite modificar los datos ingresados anteriormente en la pantalla Datos de la Clase.



7.6. Elegir Semestres para concatenación de Materias

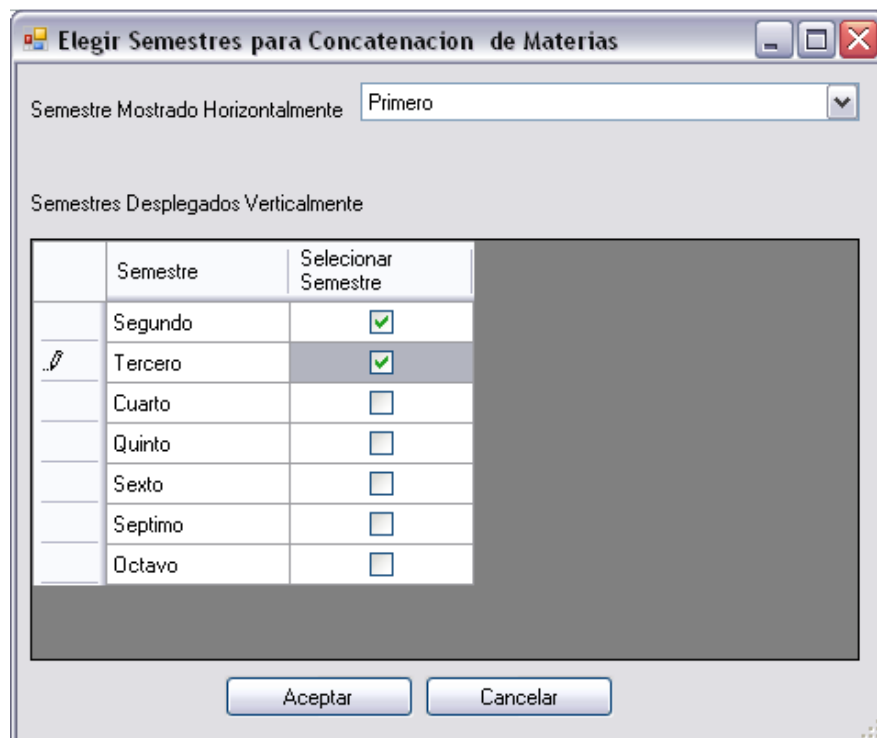


FIGURA 17 ELEGIR SEMESTRES PARA CONCATENACIÓN DE MATERIAS

En la figura 14 nos permitirá elegir semestres para concatenación de materias. Esta pantalla se encuentra en la opción herramientas y elegimos la opción “Definir Concatenación Materias”. Es recomendable definir la concatenación de materias una vez que se ha ingresado todas las materias de cada uno de los semestres, si no se encuentra ingresada por lo menos una materia el sistema desplegara un mensaje de error el cual pedirá “que se ingresen materias primero”.

En la figura 14 se establecen dos tipos de semestres: Semestres Mostrados Horizontalmente y los semestres desplegados horizontalmente los cuales mostrarán una matriz que mostraremos a continuación, el momento que se dé clic en el botón Aceptar.

La figura 15 nos muestra la concatenación de las materias del primero, segundo y tercer semestres elegidos en la figura anterior. Para establecer que una materia se encuentra concatenada con otra se da un clic en el recuadro, si el recuadro se muestra de color rojo dichas materias no están concatenadas, mientras que si se muestra de color verde como: (Física I y Física II) las dos materias se encuentran concatenadas.

	Calculo	Algebra	Fisica I	Química	Tecnologías
Traductores y	Red	Red	Red	Red	Red
Programacion	Red	Red	Red	Red	Red
Sistemas de	Red	Red	Red	Red	Red
Introduccion	Red	Red	Red	Red	Red
Arquitectura de	Red	Red	Red	Red	Red
Técnicas	Red	Red	Red	Red	Red
Fisica II	Red	Red	Verde	Red	Red
Ecología	Red	Red	Red	Verde	Red
Calculo	Verde	Red	Red	Red	Red
Ecuaciones	Red	Verde	Red	Red	Red
Matemáticas	Red	Red	Red	Red	Verde
Teoría de la	Red	Red	Red	Red	Verde

Buttons: Aceptar, Cancelar

FIGURA 18 CONCATENACIÓN DE MATERIAS

8. ADMINISTRACIÓN DE PROFESORES

8.1. Datos del Profesor

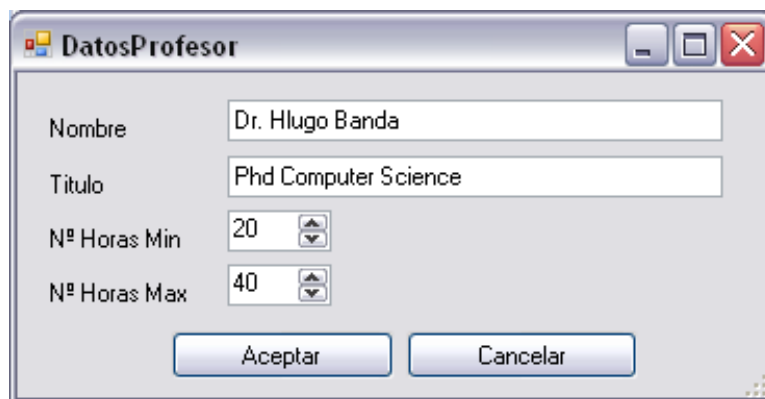
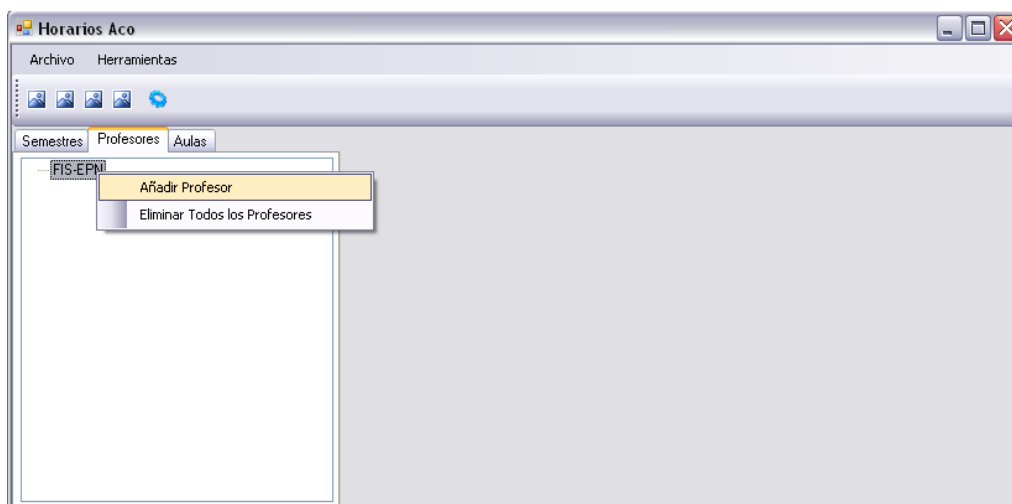


FIGURA 19 DATOS DEL PROFESOR

En la figura 16 procedemos a ingresar los datos de los profesores. Se debe ir a la opción Profesores y “Añadir Profesor” mostrados a continuación.



Una vez elegida la opción Añadir Profesor se debe ingresar los datos de la figura 16, Nombre del Profesor, Título obtenido y dos datos muy relevantes son el número máximo y mínimo de horas que el profesor labora a la semana dado que nos ayuda a optimizar de mejor manera la distribución de trabajo y las asignaciones de paralelos.

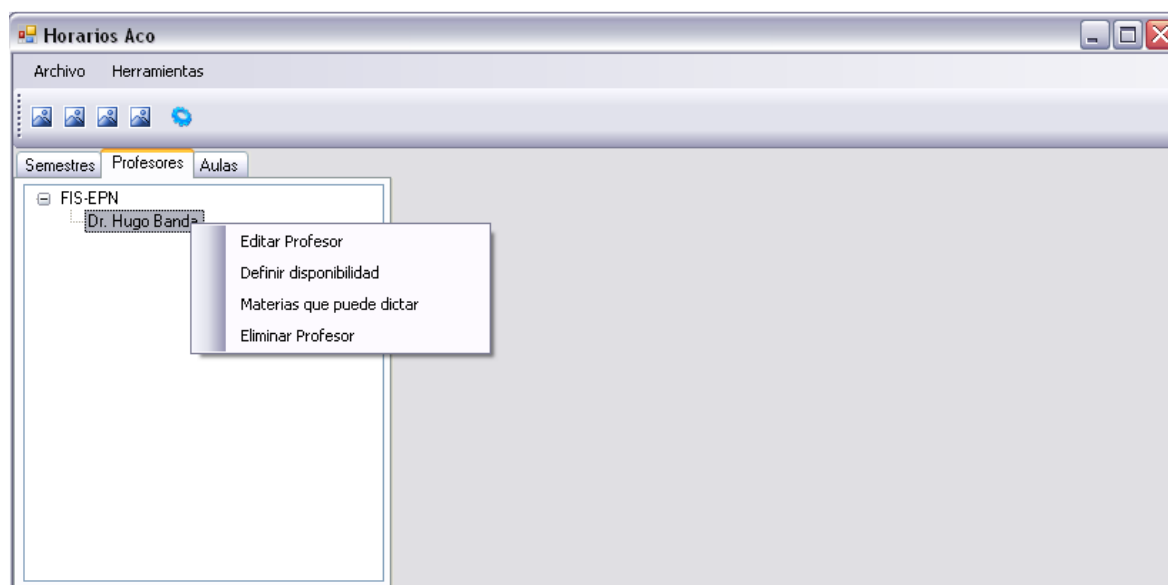
También podemos Editar el Profesor esta opción nos permite modificar los datos ingresados del profesor, para acceder a esta opción se debe dar clic derecho en el profesor ingresado y dar clic en “Editar Profesor” como se muestra a continuación.

8.2. Definir Períodos disponibles del Profesor



FIGURA 20 DEFINIR PERIODOS DISPONIBLES DEL PROFESOR

En la figura 17 se muestra la pantalla que nos permite definir los periodos disponibles del profesor, se debe ir al nombre del profesor ingresado anteriormente y a la opción “Definir disponibilidad” mostrada a continuación.



Es necesario definir en qué periodos el profesor no puede dictar clases esto se realiza usando la interfaz mostrada en la figura 17

Los periodos se deben marcar dando un clic en el recuadro, el recuadro de color rojo indican no disponibilidad y los marcados de color verde indican la disponibilidad del profesor para dictar clases.

8.3. Preferencia de las Materias que puede dictar el profesor



FIGURA 21 PREFERENCIA DE MATERIAS QUE PUEDE DICTAR EL PROFESOR

Para acceder a la pantalla de la figura 18 se debe dar clic derecho en el profesor ingresado y elegir la opción “Materias que puede dictar” como se muestra anteriormente.

Es necesario definir la Preferencia de las materias que el profesor va a dictar esto se realiza usando la interfaz mostrada en la figura 18. Una vez seleccionada si el profesor dicta o no la materia se debe establecer un número de preferencia este puede ser de 1 a 10000, a cada materia que el profesor Si puede dictar se le debe establecer un numero de preferencia.

9. ADMINISTRACIÓN DE AULAS

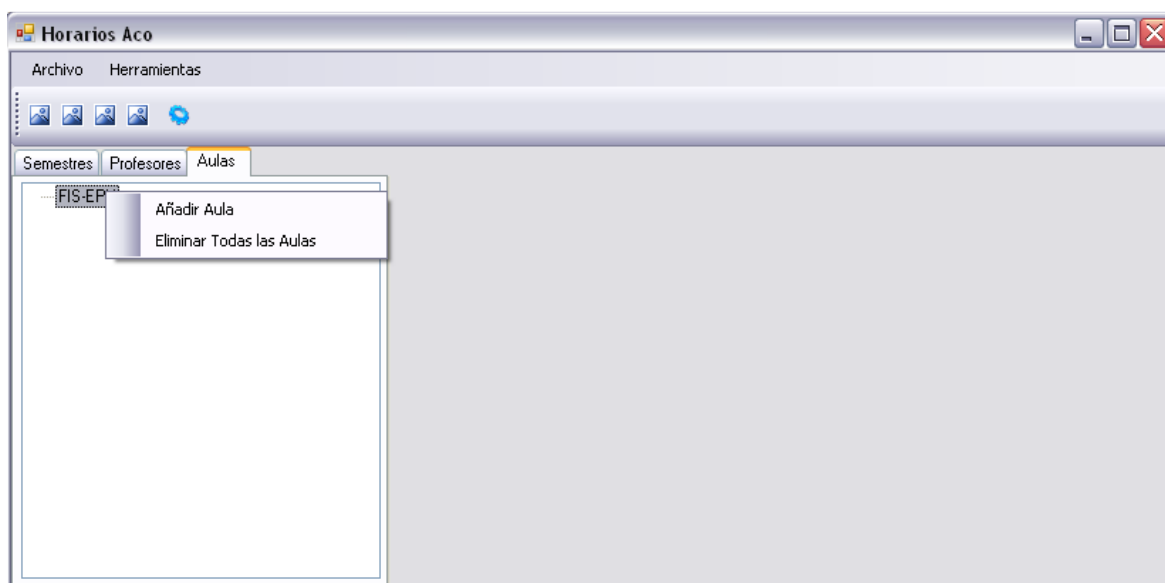
9.1. Datos del Aula



The image shows a dialog box titled "DatosAula". It has three input fields: "Nombre" containing "Aula 401", "Capacidad" containing "20", and "Tipo" containing "Aula Normal". Below these fields are two buttons: "Aceptar" and "Cancelar".

FIGURA 22 DATOS DEL AULA

En la figura 19 procedemos a ingresar los datos de las aulas. Se debe ir a la opción Aulas y “Añadir Aula” mostrados a continuación.



Es relevante también dentro del problema incluir datos del lugar donde se llevaran a cabo las clases, es decir, datos de las aulas mostradas en la figura 19 en la cual se ingresa el nombre del aula, la capacidad del aula y el tipo de aula que se definió anteriormente en la figura 6.

También se tiene la opción de Editar Aula, esta opción permite modificar los datos ingresados de las aulas, se debe dar clic derecho en una aula ingresada y luego “Editar Aula”.

9.2. Definir Períodos Disponibles de las Aulas

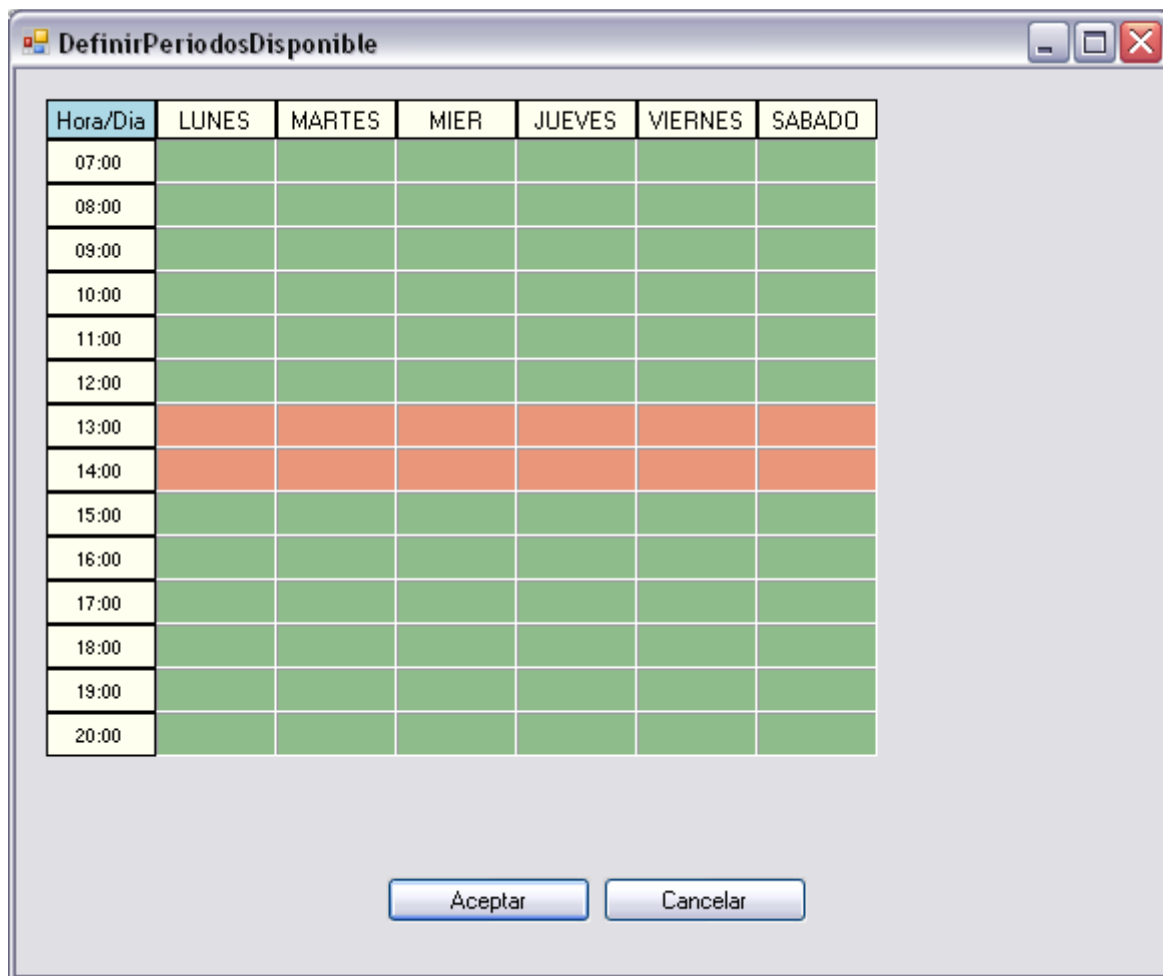


FIGURA 23 DEFINIR PERÍODOS DISPONIBLES DE LAS AULAS

Se debe definir períodos disponibles para cada aula, por lo que es necesario especificar en qué períodos el aula no estará disponible para esto se debe dar clic derecho en el aula ingresada y elegir la opción “Definir Períodos de Disponibilidad”

Para definir la disponibilidad se da clic en el recuadro, si el recuadro se encuentra de color verde el aula está disponible y si el recuadro esta de color rojo el aula no está disponible, esto se realiza usando la interfaz mostrada en la figura 20.

9.3. Definir Preferencia Aulas

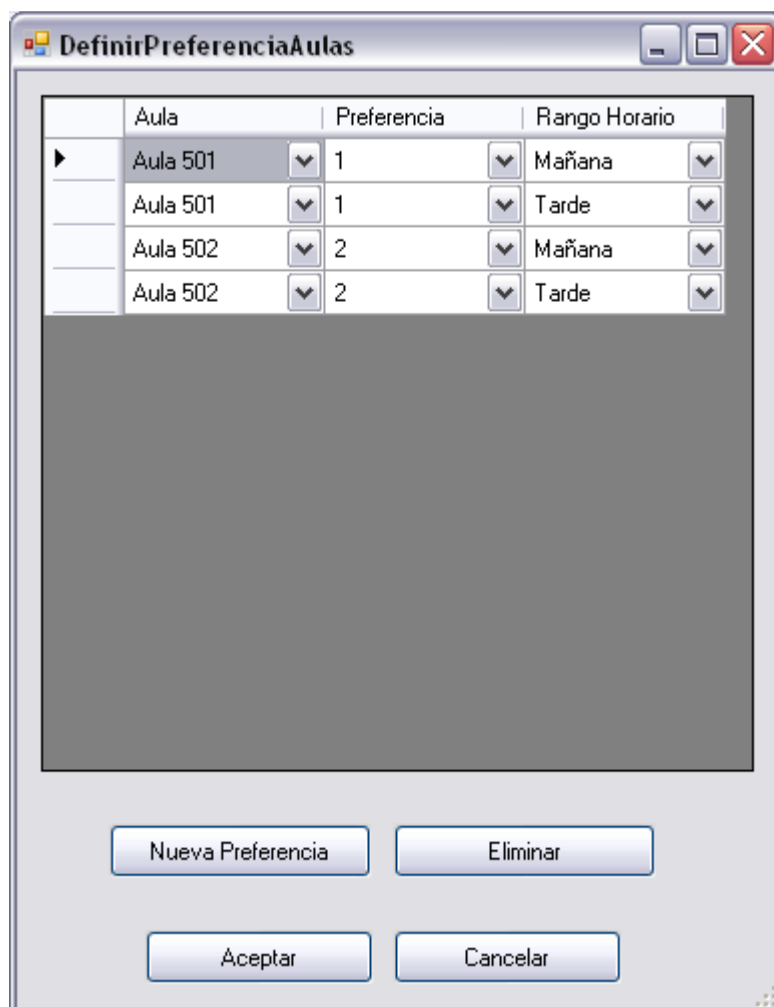


FIGURA 24 DEFINIR PREFERENCIA AULAS

En la figura 21 se define las preferencias de aulas, se debe ir a la opción de los semestres y en cada semestre dar clic derecho y la opción “Definir Preferencia Aulas”. Es recomendable ingresar a esta opción una vez que se tengan ingresadas las aulas como se muestra anteriormente. Se establece cada aula con una preferencia y un rango horario. El botón Nueva Preferencia permite añadir una nueva fila para ingresar una nueva aula con su respectiva preferencia y rango horario. El botón eliminar se debe seleccionar la fila a eliminar y dar clic en dicho botón. Los botones Aceptar y Cancelar nos permiten aceptar y cancelar la preferencia de aulas ingresadas.

10. PROCESAMIENTO DEL ALGORITMO

Una vez ingresados todos los datos necesarios, se puede realizar el proceso para obtener los resultados.

10.1. Abrir, Guardar los datos

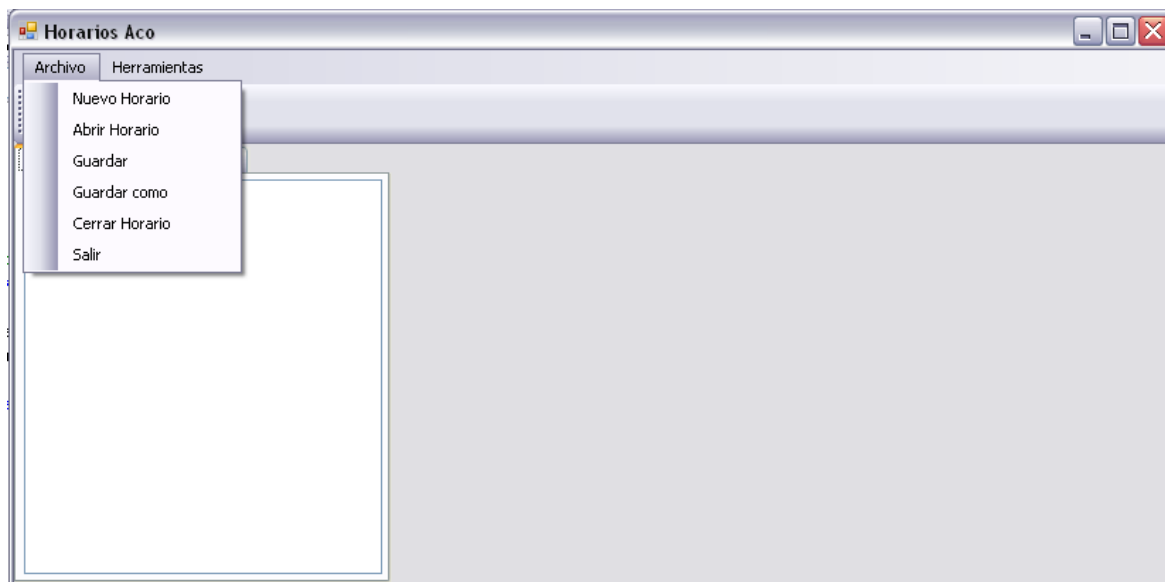


FIGURA 25 ABRIR, GURADAR LOS DATOS INGRESADOS

Una vez ingresados los datos se tiene la opción de Guardar y Guardar como estas dos opciones permiten guardar el archivo en cualquier parte de la PC. El archivo se guardará con una extensión *.aco Como se muestra a continuación.

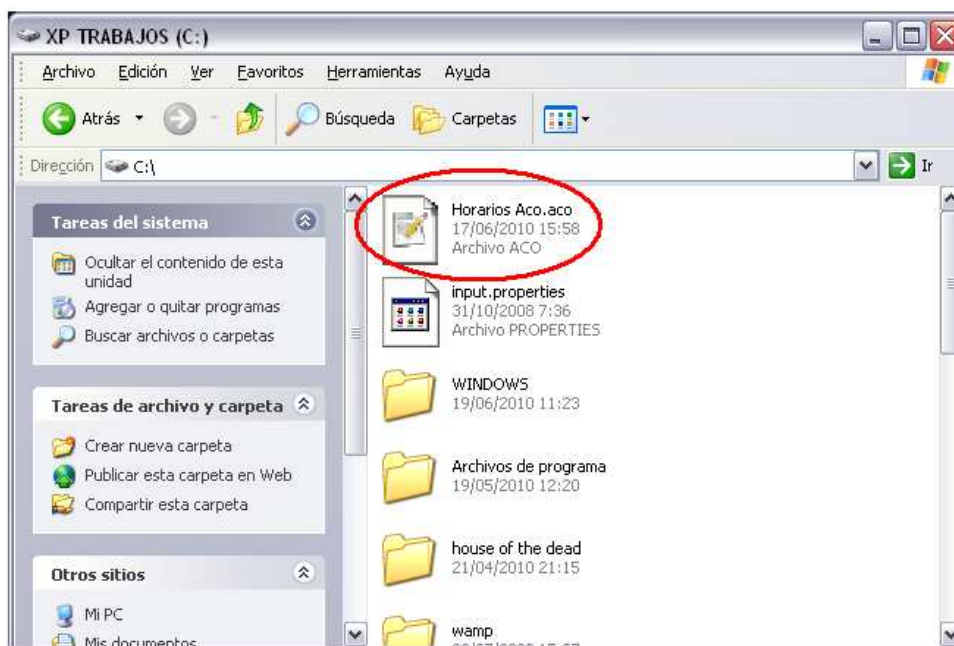


FIGURA 26 ARCHIVO GUARDADO

Se puede ir guardando los datos de acuerdo a como se vaya realizando el ingreso de los mismos.

También se tiene la opción de Abrir el archivo guardado anteriormente, se da clic en Abrir horario y los datos se volverán a cargar en la interfaz.

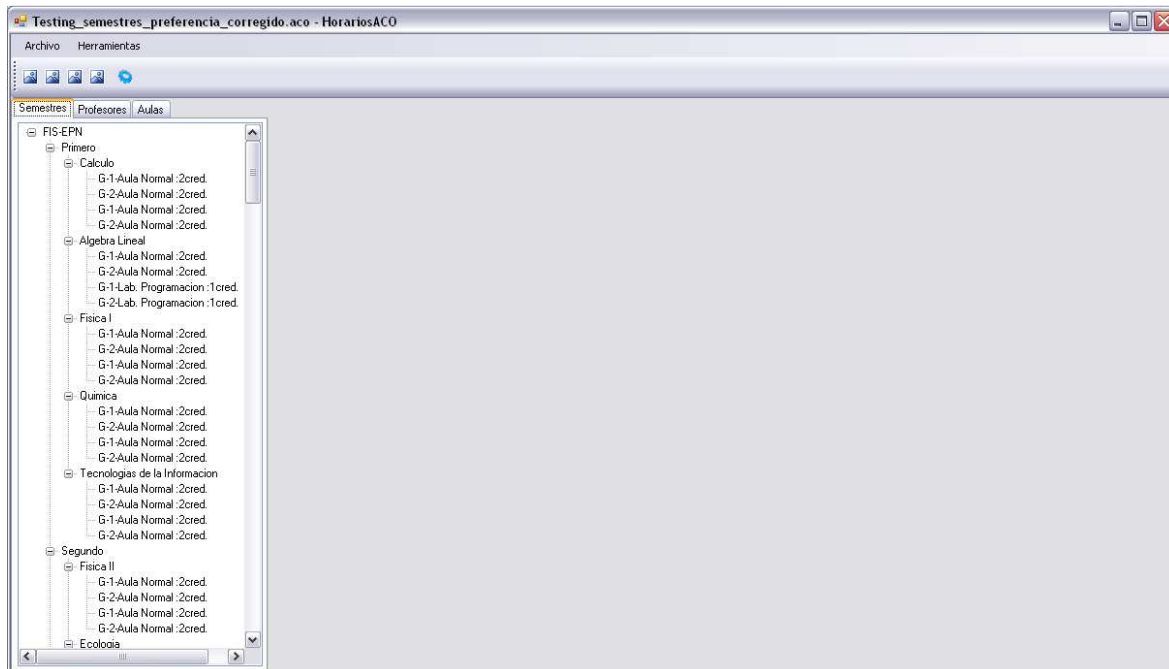


FIGURA 27 DATOS CARGADOS EN LA INTERFAZ

10.2. Proceso para obtener resultados

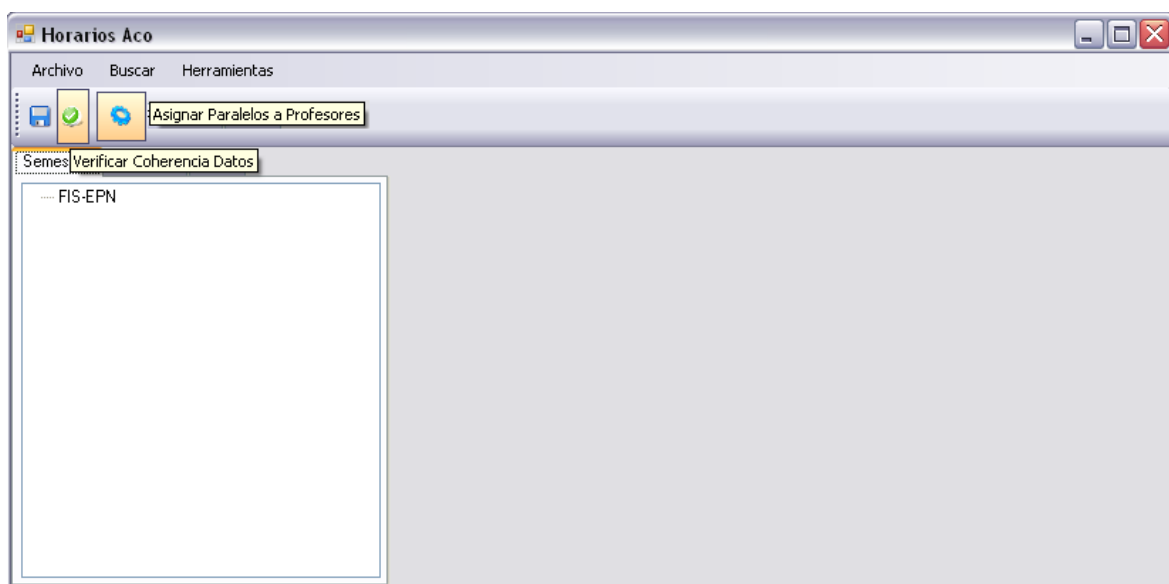


FIGURA 28 PROCESO PARA OBTENER RESULTADOS

Para obtener los resultados se debe primero verificar la coherencia de los datos es decir, se debe dar clic en el botón “Verificar coherencia de datos” mostrados en la figura 25.

Una vez que se haya realizado la verificación se procederá a ejecutar el botón “Asignar Paralelos a Profesores” mostrado en la figura 25, esto nos permitirá obtener los resultados de los horarios para la FIS de la EPN, dichos resultados se muestran en un archivo XML.

ANEXO 2

ENCUESTAS

ANEXO 3

GLOSARIO

- **Calendarización:** La Calendarización conocida como Timetabling describe una variedad de problemas de optimización bastante difíciles con considerable impacto práctico típicamente el timetabling se ocupa de la asignación de actividades a recursos.

Con más detalle, estos recursos proveen de intervalos de tiempo a los cuales las actividades se pueden asignar sujetas a ciertas restricciones laterales. El objetivo global del problema es encontrar una asignación factible de todos los eventos de tal manera que algunas propiedades deseables estén presentes en la solución final. Los problemas de timetabling son desafiantes no solo en términos de su complejidad, sino que además estos involucran varios objetivos conflictivos y también múltiples interesados con diferentes objetivos y puntos de vista

- **Meta heurística:** El sufijo “meta” significa “más allá”, a un nivel superior, las Metaheurísticas son estrategias para diseñar o mejorar los procedimientos heurísticos con miras a obtener un alto rendimiento. El término Metaheurística fue introducido por Fred Glover en 1986 y a partir de entonces han aparecido muchas propuestas de pautas o guías para diseñar mejores procedimientos de solución de problemas combinatorios. Las Metaheurísticas se sitúan conceptualmente “por encima” de las heurísticas en el sentido que guían el diseño de éstas, pueden estar compuestas por una combinación de algunas heurísticas, por ejemplo una Metaheurística puede usar una heurística constructiva para generar una solución inicial y luego usar otra heurística de búsqueda para encontrar una mejor solución.

- **Heurística:** El término heurístico está relacionado con la tarea de resolver problemas inteligentemente utilizando la información disponible, el término proviene de la palabra griega heuriskein que significa encontrar o descubrir, de la cual se deriva eureka, la famosa exclamación de Arquímedes al descubrir su principio.

En el ámbito de la Inteligencia artificial se usa el término heurístico para describir una clase de algoritmos que aplicando el conocimiento propio del problema y técnicas realizables se acercan a la solución de problemas en un tiempo razonable.

Se califica de heurístico a un procedimiento para el que se tiene un alto grado de confianza en que encuentra soluciones de alta calidad con un coste computacional razonable, aunque no se garantice su optimalidad o su factibilidad, e incluso, en algunos casos, no se llegue a establecer lo cerca que se está de dicha situación. Se usa el calificativo heurístico en contraposición a exacto.

- **Algoritmos Heurísticos:** Son algoritmos que intercambian optimidad por velocidad. Los algoritmos heurísticos no están garantizados para producir la solución óptima a un problema dado. Sin embargo, ellos pueden esperar producir una solución razonable dentro de un período de tiempo adecuado. Un método heurístico es un procedimiento para resolver un problema de optimización bien definido mediante una aproximación intuitiva, en la que la estructura del problema se utiliza de forma inteligente para obtener una buena solución.
- **Algoritmos meta heurísticos:** Son ampliamente reconocidos como uno de los enfoques más prácticos para los problemas de optimización combinatoria. Algoritmos meta heurísticos que combinan reglas y aleatoriedad imitando fenómenos naturales han sido ideados para superar los inconvenientes computacionales de algoritmos numéricos existentes. Estos algoritmos incluyen recocido simulado, búsqueda tabú y métodos de computación evolutiva.