UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE POSGRADO EN INGENIERÍA

ALGORITMO DE BÚSQUEDA TABÚ PARA UNA VARIANTE DEL PROBLEMA DE COLORACIÓN

TESIS

QUE PARA OBTENER EL GRADO DE MAESTRO EN INGENIERÍA DE SISTEMAS (IDEO)

PRESENTA:

MARIO ABOYTES OJEDA

DIRECTOR DE TESIS:
DR. JAVIER RAMÍREZ RODRÍGUEZ

COMITÉ TUTORIAL:
DRA. PATRICIA BALDERAS CAÑAS
DRA. MAYRA ELIZONDO CORTÉS

MÉXICO D.F., 2011

Indice General

Introducción

En la Investigación de Operaciones algunos problemas de planificación de eventos pueden ser modelados como problemas de coloración de gráficas, en donde los vértices representan los eventos a programar, las aristas las restricciones y los colores los recursos disponibles. Se han desarrollado una amplia gama de modelos de este tipo para la planeación de los horarios de clases en las universidades.

Un problema básico de programación de horarios de clases se puede establecer como un problema de coloración, en el cual, se colorean los vértices de la gráfica sin utilizar el mismo color en los vértices que se encuentran unidos por una misma arista. En algunas ocasiones, sin embargo, este modelo no es adecuado para plantear algún otro tipo de programación de eventos como un problema de coloración de gráficas. Este sería el caso si el tamaño del recurso a compartir es fijo y existen restricciones que exigen una determinada distribución de las clases en el horario disponible . De esta manera surge el Problema de Coloración Robusta Generalizado [12] que utiliza elementos adicionales a los que se manejan en la gráfica del problema de coloración.

El Problema de Coloración Robusta Generalizado establece una coloración robusta para una gráfica dada, con un número fijo de colores que no es necesariamente el número cromático y considera la distancia entre los colores para determinar la penalización en las aristas complementarias como función objetivo. Este problema permite plantear escenarios de programación de eventos que consideran la separación de clases en intervalos de tiempo como una restricción. Debido a que el problema se define como NP-Duro, una aproximación heurística es necesaria para encontrar buenas soluciones en una cantidad razonable de tiempo para casos extensos.

El objetivo de esta tesis es desarrollar un algoritmo heurístico utilizando gráficas para representar el problema y la metaheurística de Búsqueda Tabú para buscar soluciones factibles al problema de programación de clases con restricciones de separación de las clases en el tiempo. Esta alternativa de solución considera problemas de considerable extensión para probar su eficiencia.

En el Capítulo 1 la tesis inicia con un marco teórico del problema de programación de horarios de clases. Después en el Capítulo 2 se aborda el Problema de Coloración Robusta Generalizado propuesto por [15]. Luego en el Capítulo 3 se introducen los conceptos y definiciones necesarias para entender el funcionamiento de la Búsqueda Tabú. En el Capítulo 4 se introduce un ejemplo para mostrar la implementación del problema en una gráfica y en seguida se mencionan la heurísticas que han sido utilizadas para encontrar soluciones. En el capítulo final se describe a detalle el algoritmo de búsqueda tabú propuesto para programar los horarios de clases. Finalmente se presentan algunas conclusiones del estudio.

1. El Problema de Programación de Horarios

1.1 Introducción

La programación de horarios dentro de escuelas y universidades es una de las principales actividades administrativas para una amplia variedad de instituciones. La programación de horarios de clases o exámenes para las universidades es una tarea extensa y compleja en la que muchos departamentos y facultades diferentes han generado sus propias ideas sobre como y cuando impartir sus cursos.

En este primer capítulo se define el problema de programación de horarios de clases, los beneficios que se obtienen al resolverlo, como se hallan soluciones al problema y cuando es recomendable desarrollar un modelo para que la computadora realice el proceso de manera automática.

1.2 Calendarización de Eventos

De acuerdo a [6] los problemas de programación de eventos consisten básicamente en asignar un conjunto de clases o exámenes a un horario disponible, bajo ciertas restricciones. Dichas restricciones se dividen en dos tipos: fuertes y débiles.

Las restricciones fuertes son aquellas que deben ser satisfechas para que la planificación de los horarios sea factible. En este tipo de restricciones se encuentra por ejemplo que:

- Ningún recurso puede ser utilizado en más de dos lugares al mismo tiempo.
- Para cada intervalo de tiempo debe haber suficientes recursos (cuartos, vigilantes, etc.) disponibles para todos los eventos que hayan sido programados en dicho intervalo.

Las restricciones débiles resultan deseables pero no esenciales. Su penalización debe ser minimizada para producir la mejor solución. Ejemplos de restricciones débiles son:

- Asignación de tiempo: un curso o examen debe ser programado en un intervalo de tiempo determinado.
- Restricciones de tiempo entre eventos: un curso o examen debe ser programado antes o después de otro.
- Separación de eventos en el tiempo: los estudiantes no deben tener exámenes en periodos consecutivos o dos exámenes el mismo día.

- Coherencia: los profesores pueden preferir tener todas sus clases en un determinado número de días y tener los demás días libres de impartir alguna clase.
- Asignación de recursos: los profesores pueden preferir dar la clase en un salón en particular o puede ser el caso en el que un determinado examen deba ser programado en un cierto salón.

La amplia variedad de restricciones existentes dentro de la programación de horarios en las universidades hace que se pueda dividir en dos principales categorías: la programación de horarios de clases y la programación de horarios de exámenes.

Un número determinado de exámenes pueden ser asignados a un solo salón o un examen puede ser programado en varios salones, mientras que en la programación de clases generalmente se tiene que asignar una sola clase por salón. En la planificación de exámenes a menudo se pretende disminuir el número de exámenes próximos al estudiante, mientras que en la asignación de clases usualmente se busca el acomodo de clases consecutivas.

1.3 El Problema de Programación de Horarios de Clases

El Problema de Programación de Horarios de Clases, *PPHC* en lo que resta de la tesis, básicamente tiene un conjunto de clases $L = (l_1, l_2, \ldots, l_n)$ y un conjunto de periodos de tiempo $P = (p_1, p_2, \ldots, p_k)$ en el cual todas las clases deben ser programadas [15].

Figura 1.1 Ejemplo de un Conjunto de Clases

Curso	Materias	Hrs/Semana
Básico	Cálculo	3
	Trigonometría	3
	Estadística	3
	Matemáticas Discretas	1
Especialidad	Investigación de Operaciones	3
	Teoría de Decisiones	3
	Teoría de Inventarios	2
	Simulación	2
Complementario	Evaluación de Proyectos	2
	Tecnologías de Información	2
	Ténicas Heurísticas	2
	Cadena de Suministro	1
	Sistemas de Manufactura	1
	Ingeniería Financiera	1
	Finanzas Corporativas	1

El *PPHC* permite realizar una mejor planeación de lo recursos necesarios para poder llevar a cabo un proyecto educativo, como son el número de salones, laboratorios, escritorios, computadoras, pizarrones, proyectores, etc. Esta mejor planeación de los recursos necesarios trae consigo el beneficio implícito de reducir el costo de operación en la institución educativa.

En muchas escuelas y universidades, la programación de las clases se hace con la experiencia adquirida durante años de impartir programas educativos. Cuando se tienen nuevas ofertas educativas no resulta tan sencilla la tarea, en especial cuando el nuevo programa es bastante extenso. En la actualidad existe un incremento en la cantidad de nuevas ofertas educativas que surgen como respuesta al constante desarrollo de nuevas tecnologías que se aplican tanto en el sector privado, como en el de gobierno. Las instituciones deben contar con herramientas que les permitan tomar decisiones en la planeación de manera rápida y efectiva para poder incorporar a la brevedad estos nuevos programas en su oferta educativa.

Figura 1.2 Ejemplo de un Horario de Clases

Hr	Lunes	Martes	Miércoles	Jueves	Viernes
1	*-Estadística -Teoría de Decisiones*	*-Teoría de Inventarios -Ténicas Heurísticas*	*-Trigonometría -Teoría de Decisiones -Cadena de Suministro*	*-Teoría de Inventarios -Evaluación de Proyectos*	*-Estadística -Ingeniería Financiera*
2	*-Trigonometría -Investigación de Operaciones*	*-Simulación -Tecnologías de Información*	*-Cálculo -Investigación de Operaciones*	*-Tecnologías de Información*	*-Trigonometría -Teoría de Decisiones*
3	*-Cálculo -Evaluación de Proyectos*	*-Matemáticas Discretas -Finanzas Corporativas*	*-Estadística -Sistemas de Manufactura*	*-Simulación -Ténicas Heurísticas*	*-Cálculo -Investigación de Operaciones*

Una gran cantidad de clases a programar y la amplia variedad de restricciones impuestas en el horario hacen muy grande el conjunto de todas las posibles soluciones [5] (el espacio de búsqueda del problema). La creación del horario puede ser una tarea extremadamente difícil y la búsqueda de una solución con cálculos manuales puede requerir mucho dinero y esfuerzo.

La programación de horarios ha sido catalogada dentro de la clase de problemas conocidos como NP-completos debido a que no se conoce un método que encuentre la solución exacta al problema en una cantidad de tiempo razonable (polinomial).

Los problemas de programación de horarios han atraido la atención de la comunidad científica de una serie de disciplinas como Investigación de Operaciones e Inteligencia Artificial por cerca de 50 años y desde la década de los 90's mas o menos ha habido un interés creciente en el campo.

Una gran variedad de propuestas ha sido descrita en la literatura y probada en la realidad. Se pueden dividir en cuatro tipos: (1) métodos secuenciales, (2) métodos de agrupación, (3) métodos basados en restricciones y (4) métodos metaheurísticos. Existen propuestas híbridas que entran en más de una categoría de las que se mencionan:

1. Métodos secuenciales. Estos métodos ordenan eventos usando heurísticas de dominio y luego asignan los eventos secuencialmente en periodos de tiempo válidos para que ningún evento programado en ese momento entre en conflicto con algún otro. En los métodos secuenciales, los problemas de horarios son usualmente representados por gráficas donde las clases o exámenes son representados por los vértices, mientras que los conflictos entre los eventos son representados por las aristas.

2. Métodos de agrupación. En estos métodos, el conjunto de eventos es dividido en grupos que satisfacen las restricciones fuertes del problema y después los grupos son asignados a periodos de tiempo de tal manera que cumplan con las restricciones débiles. El principal inconveniente de estas propuestas es que las agrupaciones de los eventos son formadas y fijadas al comienzo del algoritmo lo que puede resultar en una programación de horarios de baja calidad.

3. Métodos basados en restricciones. En estos métodos un problema de programación de horarios es modelado como un conjunto de variables (eventos) a las que los valores (salones, intervalos de tiempo, etc.) tienen que ser asignados para satisfacer el número de restricciones. Usualmente se define un número de reglas para asignar los recursos a los eventos. Cuando ninguna regla aplica a la solución parcial que se tiene en ese momento se realiza un retroceso hasta encontrar una solución que satisfaga todas las restricciones.

4. Métodos metaheurísticos. Cerca de las últimas tres decadas una variedad de propuestas metaheurísticas como el recocido simulado, la busqueda tabú, los algoritmos genéticos y propuestas híbridas han sido investigadas para propósitos de programación de horarios reportando buenos resultados. Los métodos metaheurísticos comienzan con una o más soluciones iniciales y emplean estrategias de búsqueda que tratan de evitar el óptimo local. Todos estos algoritmos de búsqueda pueden producir soluciones de alta calidad pero casi siempre tienen un costo computacional considerable.

En varios trabajos que utilizan metaheurísticas y propuestas híbridas para la resolución de diversos problemas se han obtenido buenos resultados como en [17] que muestra una diversa aplicación del recocido simulado de manera exitosa. El *PPHC* puede ser modelado como un problema de coloración de gráficas para facilitar la incorporación de modelos heurísticos a su resolución [3,4] creando un algoritmo híbrido. Esta propuesta ha dado resultados aceptables en estudios anteriores como el mostrado en [15].

El modelo de coloración mínima se ha utilizado en la programación de horarios de exámenes [12] obteniendo buenos resultados, sin embargo, el *PPHC* que se aborda en este estudio no puede ser resuelto utilizando este enfoque.

El problema que se define en este trabajo tiene restricciones de separación de clases en el tiempo, las cuales son:

- Dos clases de un mismo curso deben ser asignadas en distintas horas.
- Dos clases de una misma materia no pueden ser programadas en el mismo día ni en días consecutivos.
- No pueden programarse más de cierto número de clases por hora dado el número limitado de espacios.

J. Ramírez (2001) [12] desarrolló un problema de coloración de gráficas que hace posible incluir las restricciones de separación de eventos en el tiempo que se mencionan. El Problema de Coloración Robusta Generalizado permite modelar el *PPHC* como una gráfica para despúes buscar una solución al problema mediante alguna heurística.

P. Lara et al (2010) [15] utilizan el Problema de Coloración Robusta Generalizado para modelar el *PPHC* y mediante un algoritmo GRASP (Greedy Randomized Adaptive Search Procedure en inglés) resolver instancias con restricciones de separación de eventos en el tiempo, obteniendo buenos resultados.

En el siguiente capítulo se presenta el Problema de Coloración Robusta Generalizado. El capítulo comienza introduciendo el Problema de Coloración de Vértices para después continuar con el desarrollo del Problema de Coloración Mínima, en seguida se muestra el Problema de Coloración Robusta y se finaliza con el modelo que hace posible estructurar el problema como una gráfica.

2. La Teoría de Gráficas

2.1 Introducción

Dentro de la Investigación de Operaciones se encuentra el campo de conocimiento denominado Teoría de Gráficas, que se encarga de estudiar las estructuras matemáticas llamadas gráficas. Dichas estructuras se utilizan para modelar relaciones uno a uno entre objetos de una colección determinada.

La Teoría de Gráficas ha proporcionado muchos modelos y técnicas de solución eficientes aplicados a una gran variedad de problemas que han surgido en diferentes contextos. Uno de los problemas que se ha resuelto con buenos resultados mediante el uso de las gráficas es el *PPHC*.

En este capítulo se introduce el Problema de Coloración de Vértices en una gráfica y después se define el Problema de Coloración Mínima. Estos dos problemas dan las bases para abordar el Problema de Coloración Robusta y finalmente el Problema de Coloración Robusta Generalizado que se utiliza para modelar el problema que se plantea en esta tesis.

2.2 El Problema de Coloración Mínima

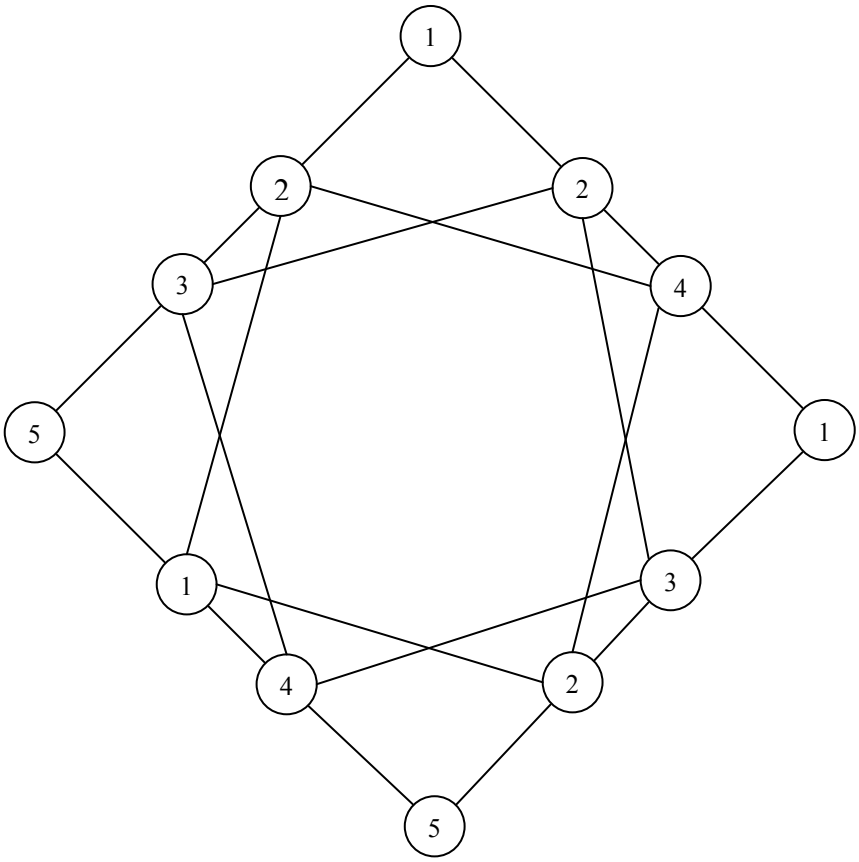
Uno de los principales problemas que se encuentran dentro de la Teoría de Gráficas es el problema de coloración de vértices de una gráfica que consiste en asignar un color, representado por un número entero positivo, a cada vértice de tal manera que cualquier par de nodos adyacentes no tengan el mismo color.

De a cuerdo a [12] dada una gráfica $G=(V,A)$, con $|V|=n$, se define una función de coloración como una asignación $C:\{1,...,n\}\rightarrow\{1,2,...\}$ que identifica a $C(i)$ como el color del vértice $i\in V$ de forma que dos vértices adyacentes, $\{i,j\}\in A$ no tienen el mismo color, es decir, $C(i)\neq C(j)$.

Dada una función de coloración C de una gráfica G, al conjunto de vértices que tienen asignado el mismo color se le nombra clase de color: $V_c(k)=\{i\in V|C(i)=k\}\quad\forall k\in\{1,2,...\}$. Una *k-coloración* es una función de coloración que no utiliza más de *k* colores: $C^k:\{1,...,n\}\rightarrow\{1,2,...,k\}$. Una gráfica es *k-coloreable* si admite una *k-coloración*.

En la Figura 2.1 se puede observar el Problema de Coloración. Ninguno de los vértices de la gráfica que se encuentran unidos por las aristas están coloreados con el mismo color.

Figura 2.1 Problema de Coloración de Vértices



Al mínimo valor *k* tal que G es *k-coloreable* se le llama *número cromático* de la gráfica y se denota por $\chi(G)$. Dada una gráfica G, el Problema de Coloración Mínima, denotado por *PCM* en lo que resta de la tesis, busca una función de coloración que no utilice más de $\chi(G)$ colores.

La programación de exámenes, la determinación de conglomerados, la coloración de un mapa, etc son problemas que pueden ser modelados como problemas de coloración para encontrar los recursos óptimos necesarios utilizando el número cromático de la gráfica.

El *PCM* se puede modelar con la programación entera, la programación binaria, el modelo de partición, el modelo de asignación, el modelo de asignación cuadrática, etc. En este trabajo se utiliza el modelo de programación binaria en todos los problemas para poder comparar sus características e identificar sus diferencias significativas.

En la siguiente sección se presenta el modelo introduciendo las variables de decisión binarias, las variables endógenas, la función objetivo y sus respectivas restricciones. En el trabajo de J. Ramírez (2001) [12] se muestran otro modelos como el modelo de programación entera, el modelo de partición, el modelo de asignación y el modelo de asignación cuadrática.

2.2.1 Programación de la Coloración Mínima

Para el desarrollo del modelo se tienen las variables de decisión binarias:

$$x_{ik} = \begin{cases} 1 \; si \; C(i) = k \\ 0 \; si \; C(i) \neq k \end{cases} \quad \forall k \in \{1,\ldots,n\}$$

Y las variables endógenas:

$$y_k = \begin{cases} 1 \; si \; el \; color \; k \; es \; usado \\ 0 \; si \; el \; color \; k \; no \; es \; usado \end{cases} \quad \forall k \in \{1,\ldots,n\}$$

El modelo de programación es entonces:

$$min \sum_{k=1}^{n} k \cdot y_k$$

$$s.a. \sum_{k=1}^{n} x_{ik} = 1, \quad \forall i \in V$$
$$x_{ik} + x_{jk} \leq 1, \quad \forall \{i, j\} \in A$$
$$\sum_{i=1}^{n} x_{ik} \leq n \cdot y_k, \quad \forall k \in \{1,\ldots,n\}$$

$$y_k, x_{ik} \in \{0,1\}, \quad \forall i \in V, k \in \{1,\ldots,n\}$$

La función objetivo se define de esta manera para asegurar que se agoten antes los colores más bajos, es decir, empezando desde el número más pequeño. La primera restricción del modelo asegura que se le de un color a cada vétice, la segunda restricción previene la incompatibilidad de colores en los extremos de las aristas y la tercera garantiza que si se utiliza algún color la variable y_k tenga un valor de 1 y cuente en la función objetivo.

El *PCM* es un ejemplo básico dentro de la Teoría de Gráficas y aunque tiene una amplia gama de aplicación como modelo para diversos problemas, entre los que se encuentran la programación de horarios de exámenes, tiene varias limitantes cuando las restricciones adquieren una mayor complejidad en el problema.

En el caso de la programación de horarios en donde las restricciones establecen una determinada separación en el tiempo de los eventos, el problema de coloración mínima resulta insuficiente para modelar el problema. En la siguiente sección se introduce el Problema de Coloración Robusta para después abordar el Problema de Coloración Robusta Generalizado que permite incluir las restricciones temporales.

2.3 El Problema de Coloración Robusta

Según [12] algunos problemas de planificación del tiempo o de recursos se pueden plantear como problemas de coloración de los vértices de una gráfica, intentando utilizar el mínimo número de colores posible. Los vértices representan los elementos a programar y cada arista identifica la incompatibilidad entre los elementos correspondientes. Esta incompatibilidad representa el conflicto entre recursos cuando no puedan ser compartidos los elementos a programar.

Una coloración válida representa una asignación adecuada de recursos compatibles. En estos problemas es común que el objetivo sea utilizar el mínimo número de colores, de tal forma que a cada par de elementos a programar que no pueden compartir el mismo recurso se les asigne un color distinto. Cada clase de color contendrá los elementos entre los que no hay incompatibilidades y que pueden compartir el recurso que se encuentra disponible en ese momento. En ocasiones el recurso a minimizar, con el planteamiento del mínimo número de colores, no es fundamental. En su lugar lo que interesa es que una solución al problema sea estable, en el sentido de que al añadir o cambiar aristas en la gráfica, la coloración de la misma continúe siendo válida. Este tipo de consideraciones muestran que el problema de coloración mínima es un modelo muy restrictivo para este tipo de situaciones.

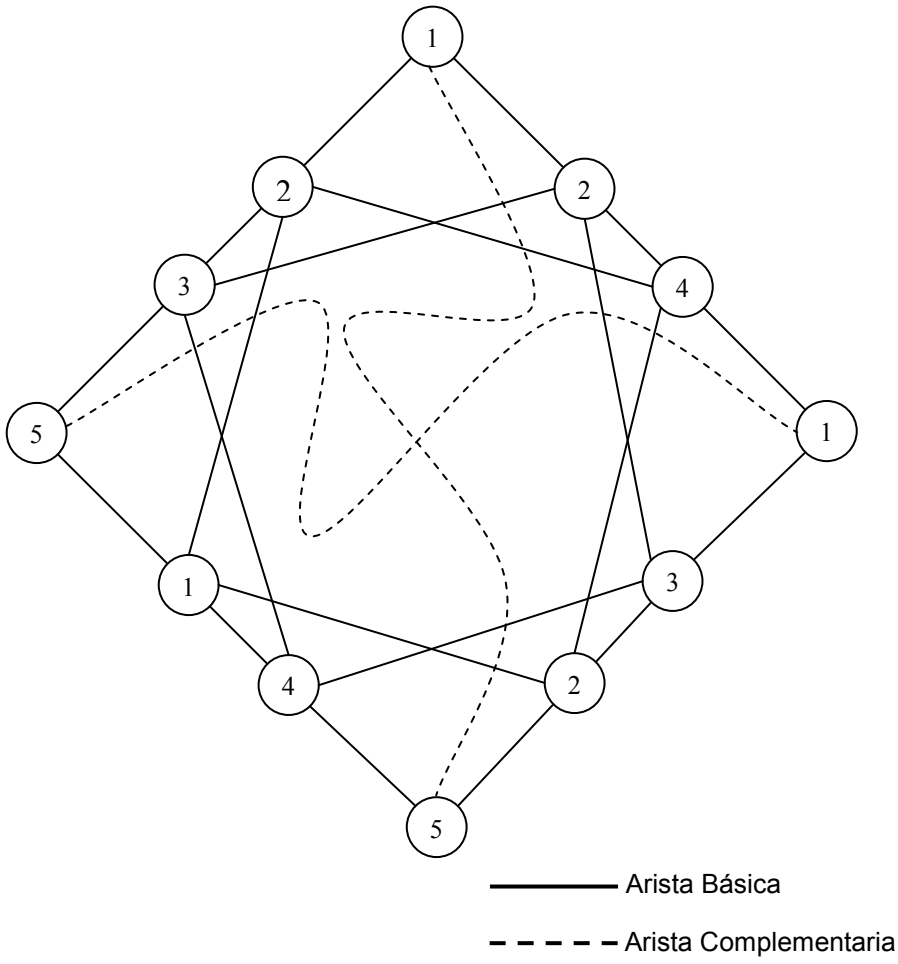
En esta sección se presenta un problema de coloración de gráficas que contempla las extensiones mencionadas. En este problema, cada planificación se hará teniendo en cuenta un determinado número de recursos, diversas programaciones alternativas, coloraciones de la gráfica asociada más flexibles y que puedan ser evaluadas con otros criterios.

El Problema de Coloración Robusta, denominado *PCR* en el resto de la tesis, valora la coloración de una gráfica en la medida en que siga siendo una coloración válida al añadir nuevas aristas. El *PCR* tiene un conjunto de aristas denominadas complementarias que no se contemplan en el *PCM*. El conjunto de aristas complementarias $\overline{A}$, representan las aristas que se podrían agregar a la topología de la gráfica. Si la coloración de una gráfica se conserva como una solución válida después de incluir las aristas complementarias en la estructura, se dice entonces que se tiene una solución robusta al cambio de las condiciones originales del problema.

El nivel de rigidez de la gráfica es inversamente proporcional al grado de robustes de la misma, por tanto, a mayor robustes en la coloración se tiene un menor nivel de rigidez en la solución.

En la Figura 2.2 se muestra una gráfica con aristas complementarias que se pueden agregar a su topología. Si la coloración sigue siendo válida después del cambio entonces la gráfica tiene una coloración robusta. En el *PCR* se busca la coloración con el menor nivel de rigidez.

Figura 2.2 Problema de Coloración Robusta



Entonces dada una gráfica $G=(V,A)$, un número de colores válido $c>0$, y una familia de penalizaciones de las aristas complementarias $\{p_{ij},\ \{i,j\}\in\overline{A}\}$, el Problema de Coloración Robusta consiste en determinar aquella *c*-coloración C_R^c con menor grado de rigidez:

$$R\left(C_R^c\right)=\min_{C^c} R\left(C^c\right)$$

Considerando que la matriz de adyacencia de la gráfica G es simétrica y que se conocen las penalizaciones p_{ij} de las aristas complementarias $\{i,j\}\in\overline{A}$, cada ejemplo del *PCR* queda caracterizado por los parámetros n, número de vértices de la gráfica; c, el número de colores permitidos; y la matriz cuadrada H de dimensión $n\times n$ que almacena en la matriz triangular inferior la matriz de adyacencia de G y en la matriz triangular superior las penalizaciones:

$$h_{ij}=\begin{cases} p_{ij} & si\ i<j \\ b_{ij} & si\ i>j \end{cases}$$

De tal forma que la matriz H es:

$$\begin{pmatrix} 0 & p_{12} & p_{13} & \cdots & p_{1j} & \cdots & p_{1n} \\ b_{12} & 0 & p_{23} & \cdots & p_{2j} & \cdots & p_{2n} \\ b_{13} & b_{23} & 0 & \cdots & p_{3j} & \cdots & p_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \cdots & \vdots \\ b_{1j} & b_{2j} & b_{3j} & \cdots & 0 & \cdots & p_{jn} \\ \vdots & \vdots & \vdots & \cdots & \vdots & \ddots & \vdots \\ b_{1n} & b_{2n} & b_{3n} & \cdots & b_{jn} & \cdots & 0 \end{pmatrix}$$

B es la matriz de adyacencia:

$$b_{ij} = \begin{cases} 1 & si\ \{i,j\} \in A \\ 0 & si\ \{i,j\} \notin A \end{cases} \quad i < j$$

Y p_{ij} es la penalización del par $\{i,j\} \notin A$ que vale 0 si $\{i,j\} \in A$. La penalización está directamente relacionada con la probabilidad de agregar una arista complementaria a la gráfica original. En la siguiente sección se define el modelo de programación matemática del problema.

2.3.1 Programación de la Coloración Robusta

Sean variables de decisión

$$x_{ik} = \begin{cases} 1 & si\ C(i) = k \\ 0 & si\ C(i) \neq k \end{cases} \quad \forall i \in \{1,\ldots,n\} \quad \forall k \in \{1,\ldots,c\}$$

Se introducen las siguientes variables auxiliares

$$y_{ij} = \begin{cases} 1 & si\ existe\ k \in \{1,\ldots,c\}\ tal\ que\ x_{ik} = x_{jk} \\ 0 & en\ caso\ contrario \end{cases} \quad \forall \{i,j\} \in \overline{A}$$

Para plantear el *PCR* como un problema de programación binaria:

$$min \sum_{\{i,j\} \in \overline{A}} p_{ij} y_{ij}$$

$$\begin{aligned} &s.a. \sum_{k=1}^{c} x_{ik} = 1 && \forall i \in \{1,\ldots,n\} \\ &x_{ik} + x_{jk} \leq 1, && \forall \{i,j\} \in A\ y\ \forall k \in \{1,\ldots,c\} \\ &x_{ik} + x_{jk} - 1 \leq y_{ij} && \forall \{i,j\} \in \overline{A}\ y\ \forall k \in \{1,\ldots,c\} \end{aligned}$$

La primera familia de restricciones, al igual que en la formulación para el *PCM*, asegura que a cada vértice se le asigne un solo color. La segunda familia garantiza que la coloración sea válida y la tercera garantiza que si dos vértices $i, j \in V$ no unidos por una arista tienen el mismo color, entonces la variable auxiliar y_{ij} vale 1.

Aunque el *PCR* es un problema que permite modelar problemas adicionales a los que aborda el *PCM*, no es posible utilizarlo en el *PPHC* con restricciones de separación de las clases en el tiempo. Sin embargo el Problema de Coloración Robusta Generalizado que se presenta en la siguiente sección, mantiene una fuerte relación con el *PCR* y tiene una estructura que le permite abordar este tipo de problema.

2.4 El Problema de Coloración Robusta Generalizado

Con un conjunto de recursos limitados o escasos y una serie de eventos determinados, como pueden ser las clases de un programa escolar, surge el problema de asignar dichos eventos a un horario establecido. Se forma así una gráfica donde los vértices son los elementos a planificar y las aristas unen los elementos que no pueden compartir el mismo recurso. La coloración en las aristas identifica el recurso asignado (como puede ser una hora determinada dentro de un horario).

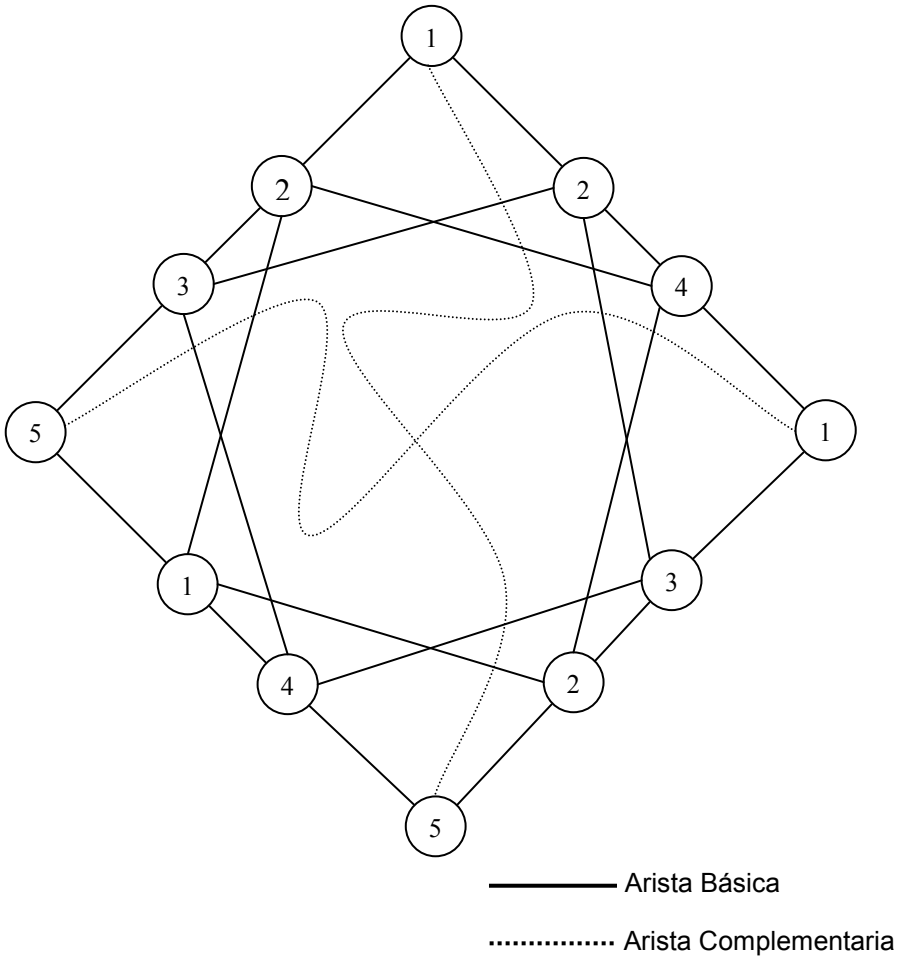
Según se menciona en [12] puede ocurrir además, que la programación de los eventos esté limitada por restricciones temporales del tipo “dos eventos no pueden ser programados en el mismo día” o “debe haber al menos dos días entre dos eventos de un mismo tipo”; también con restricciones de espacio como “no pueden programarse más de *k* eventos en cada hora”, por disponer de un número limitado de salones por ejemplo. En estos casos, el *PCR* no resuelve el problema.

El Problema de Coloración Robusta Generalizado, al cual llamaremos *PCRG* en el resto de la tesis, considera una gráfica G con un conjunto de vértices V, un conjunto de aristas A y un conjunto de aristas complementarias $\overline{A}$. En el *PCRG* ambos conjuntos de aristas existen en la gráfica.

El *PCRG* introduce la definición de distancia entre los colores que se define para cada par de vértices unido por una arista complementaria. En la siguiente sección se trata a detalle el concepto de distancia entre colores y la matriz M que proporciona una mayor flexibilidad en el manejo de las distancias requeridas entre los colores mencionados.

La Figura 2.3 muestra un problema en donde los colores existentes en los vértices de sus aristas complementarias deben tener una separación de acuerdo al problema que se haya planteado (Una distancia de 6 por ejemplo). Las aristas básicas del problema únicamente establecen que la coloración en sus extremos no sea la misma.

Figura 2.3 Problema de Coloración Robusta Generalizado



Con el fin de definir la diferencia entre los colores, se utiliza una distancia entre los mismos. Se utiliza el concepto de distancia que se introduce en el trabajo de Ramírez (2001) [12]. Dado un conjunto no vacío *C*, se define una distancia como una aplicación: $d : \mathrm{C} \times \mathrm{C} \Rightarrow \Re$

Verificando las siguientes propiedades:

1. Simetría:

$$d(x,y) = d(y,x) \quad \forall x, y \in C$$

2. No Negatividad:

$$d(x,y) \geq 0 \quad \forall x, y \in C$$

3. Identidad:

$$d(x,x) = 0 \quad \forall x \in C$$

Dada una *c*-coloración C^c, se define una <u>*c*-distancia</u> d como una distancia en el conjunto de colores $\{1,2,\ldots,c\}$. Un ejemplo de *c*-distancia es la trivial:

$$d^0(k,k') = \begin{cases} 0 & k = k' \\ 1 & k \neq k' \end{cases} \quad \forall k,k' \in \{1,2,\ldots,c\}$$

Otra *c*-distancia puede ser la basada en el valor absoluto de la diferencia de los colores:

$$d^1(k,k') = |k-k'| \quad \forall k,k' \in \{1,2,\ldots,c\}$$

Como se puede ver en [12] a partir de una *c*-distancia d y de un umbral $\hat{d} \geq 0$ se define el grado de rigidez generalizado. Fijada una *c*-distancia de colores d, a partir del grafo G y conocida la familia de penalizaciones $\{p_{ij} \geq 0, \{i,j\} \in \overline{A}\}$, se define el $\hat{d}$-grado de rigidez generalizado de la *c*-coloración C^c, siendo $\hat{d} \geq 0$, a la suma de las penalizaciones de los aristas complementarios cuyos extremos están coloreados con colores que tienen una distancia menor o igual a $\hat{d}$. Se denota por $R^{\hat{d}}(C^c)$:

$$R^{\hat{d}}(C^c) \equiv \sum_{\{i,j\} \in \overline{A}, d(C^c(i), C^c(j)) \leq \hat{d}} p_{ij}$$

El *PCR* es un caso particular de *PCRG*, en el que la *c*-distancia es d^0 y $\hat{d} = 0$.

2.4.1 Programación de la Coloración Robusta Generalizada

Con el fin de incorporar la *c*-distancia entre colores

$$d : \{1,2,\ldots,c\} \times \{1,2,\ldots,c\} \to \{0,1,2,\ldots\}$$

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

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

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

$$M = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

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

$$M = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \qquad M' = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

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

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

$$x_{ik} = \begin{cases} 1 \; C^c(i) = k \\ 0 \; C^c(i) \neq k \end{cases} \quad i \in \{1, \ldots, n\}, k \in \{1, \ldots, c\}$$

Las restricciones del modelo de programación son las siguientes:

- La coloración debe estar bien definida en todos los vértices

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

- La coloración es válida

$$x_{ik} + x_{jk} \leq 1 \qquad \forall \{i, j\} \in A \quad \forall k \in \{1, \ldots, c\}$$

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

$$y_{ij} = \begin{cases} 1 \quad si \; d\left(C^c(i), C^c(j)\right) \leq \hat{d} \quad \{i, j\} \in \overline{A} \\ 0 \quad si \; d\left(C^c(i), C^c(j)\right) > \hat{d} \quad \{i, j\} \in \overline{A} \end{cases} \quad \forall \{i.j\} \in \overline{A}$$

Es decir, $y_{ij}=1$ si y sólo si los extremos de la arista complementaria $\{i,j\}\in\overline{A}$, con colores $k=C^c(i)$ y $k'=C^c(j)$, verifican $m_{kk'}=0$. Con el fin de garantizar esta propiedad se introduce la siguiente familia de restricciones lineales:

$$x_{ik}+\sum_{k'=1}^{c}x_{jk'}(1-m_{kk'})\le y_{ij}+1 \qquad \forall\{i,j\}\in\overline{A},\ \forall k\in\{1,\ldots,c\}$$

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

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

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

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

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

El *PCRG* es muy flexible porque permite definir diferentes distancias entre colores. En la programación semanal la asignación de las primeras horas de clase del día, colores 1,2,3,..., pueden ser incompatibles con las últimas horas del día anterior colores,..., $c-2$, $c-1$, c, por lo tanto, una d-distancia circular determinaría, para un límite establecido, una matriz de tipo:

$$M=\begin{pmatrix}0&0&0&\cdots&1&0&0\\0&0&0&\cdots&1&1&0\\0&0&0&\cdots&1&1&1\\&\cdots&&\cdots&&\cdots&\\1&1&1&\cdots&0&0&0\\0&1&1&\cdots&0&0&0\\0&0&1&\cdots&0&0&0\end{pmatrix}$$

La Matriz M permite identificar situaciones más generales que aquellas consideradas por una d-distancia definida en el conjunto de colores y un límite o umbral $\hat{d}$.

Una vez que se han introducido conceptos acerca del *PCRG*, los cuales son fundamentales para el desarrollo de este trabajo de tesis, en el siguiente capítulo se introduce brevemente la técnica metaheurística que se denomina Búsqueda Tabú con la definición de sus conceptos básicos para entender su funcionamiento, como son los movimientos de permutación, la memoria tabú, etc. Estas definiciones permitirán entender la implementación de la técnica al algoritmo de programación de horarios de clases.

3. La Búsqueda Tabú

3.1 Introducción

El tercer capítulo del presente estudio introduce el marco teórico de la Búsqueda Tabú, dando los conceptos básicos necesarios para poder diseñar un algoritmo que resuelva el *PPHC* partiendo de su modelo de gráfica. Este capítulo inicia introduciendo definiciones como búsqueda de entorno, historial de búsqueda, dimensiones de memoria, etc.

La búsqueda tabú (*BT* en lo que resta de la tesis) es un procedimiento heurístico que se distingue por el uso de una memoria adaptativa y de estrategias especializadas de procesamiento de información, fue desarrollado por [7,8] y varios investigadores la han utilizado para abordar una amplia gama de problemas. De acuerdo a [9] la memoria adaptativa del algoritmo hace uso de la historia en el proceso de búsqueda de la mejor solución al problema haciendo referencia a cuatro dimensiones principales, basadas en lo reciente, en la frecuencia, la calidad y la influencia.

La *BT* tiene sus orígenes en metodologías diseñadas para salir de óptimos locales y explorar nuevas regiones en el espacio de soluciones. El éxito relevante del método para problemas de optimización duros ha causado un brote considerable de nuevas aplicaciones en los últimos años. Una de las aplicaciones de la *BT* es en problemas que realizan operaciones de permutación o intercambio de pares para buscar optimizar la función objetivo.

Los problemas de permutación o intercambio son un tipo muy importante en la parte de optimización, pues contienen una vasta literatura al respecto, como son problemas del agente viajero, asignación cuadrática, secuenciación de la producción y una variedad de problemas de diseño entre otros. Algunos de ellos han sido estructurados para adaptar la búsqueda tabú en sus respectivos espacios de solución con éxito. En este estudio se plantea el *PCRG* como una instancia de permutación para facilitar su diseño.

En [9] mencionan que los procedimientos de *BT* trabajan bajo la suposición de que se puede crear un entorno para identificar soluciones vecinas que puedan ser alcanzadas desde una posición inicial en el espacio de búsqueda. Las permutaciones son comúnmente utilizadas para definir dominios en ejercicios de optimización combinatoria, identificando movimientos que llevan de una solución a la siguiente, explorando el entorno de búsqueda.

La búsqueda tabú puede interpretarse como una búsqueda en el entorno, aunque aquí tiene un significado más complejo. La *BT* por ejemplo incluye procedimientos constructivos y destructivos que dirige por memoria adaptativa, mientras que dichos procedimientos no son incluidos en la búsqueda básica en el entorno.

3.2 La Memoria de la Búsqueda Tabú

Según [7], una representación adecuada de búsqueda en el entorno define para cada solución $x \in X$, un conjunto asociado de vecinos, $N(x) \subset X$, llamado entorno o vecindad de x. En *BT*, los entornos normalmente se asumen simétricos, es decir, x' es un vecino de x sí y sólo si x es un vecino de x'. Los pasos para el método de búsqueda en el entorno de acuerdo a [7] se muestran en la Figura 3.1.

Figura 3.1 Método de Búsqueda en el Entorno

Paso 1 (inicialización)

(A) Seleccionar una solución de arranque $xActual \in X$.

(B) Almacenar la mejor solución actual conocida haciendo $xMejor = xActual$ y definiendo $MejorCosto = c(xMejor)$.

Paso 2 (Elección y finalización)

Elegir una solución $xSiguiente \in N(xActual)$. Si los criterios de elección empleados no pueden ser satisfechos por ningún miembro de $N(xActual)$, o si se aplican otros criterios de parada, entonces el método para.

Paso 3 (Actualización)

Rehacer $xActual = xSiguiente$, y si $c(xActual) < MejorCosto$, ejecutar el paso 1(B). Volver al paso 2.

La *BT* emplea una metodología diferente al método de búsqueda en el entorno para ir más allá del criterio de finalizar en un óptimo local. Se reduce el uso de la aleatorización, para utilizar una búsqueda inteligente basada en formas sistemáticas de dirección como la memoria adaptativa.

El efecto de la memoria puede ser previsto estipulando que *BT* de acuerdo a [7] mantiene una historia selectiva H de los puntos en el espacio encontrados durante la búsqueda, y reemplaza $N(xActual)$ por un entorno modificado que puede ser denotado $N(H, xActual)$. Entonces, la historia determina que soluciones pueden ser alcanzadas por un movimiento desde la solución actual, seleccionando $xSiguiente$ de $N(H, xActual)$.

La *BT* hace uso de la historia para crear una evaluación modificada de las soluciones que se pueden acceder desde una solución actual, reemplazando $c(x)$ por $c(H, x)$. Con esta función se evalúa la calidad de las soluciones propuestas.

Según [8] esta función modificada es relevante porque *BT* usa criterios de decisión agresivos que buscan un mejor $xSiguiente$, es decir, que proporcionan un mejor valor de $c(H, xSiguiente)$, sobre un conjunto candidato trazado de $N(H, xAhora)$. Además, las evaluaciones modificadas están en ocasiones acompañadas de la alteración sistemática de $N(H, xActual)$, para incluir soluciones vecinas que no satisfacen las condiciones de factibilidad. La referencia a $c(x)$ y a la factibilidad se mantiene para determinar si un movimiento es de mejora o conduce a una nueva mejor solución.

Para problemas grandes, donde $N(H, xActual)$ puede tener muchos elementos, o para problemas donde estos elementos pueden ser costosos de examinar, la orientación de elección agresiva de *BT* hace altamente importante aislar un subconjunto candidato del entorno, y examinar este subconjunto en vez del entorno completo. Esto puede realizarse en etapas, permitiendo que el subconjunto candidato se extienda si no se encuentran alternativas que satisfagan los niveles de aspiración.

Debido a la importancia del papel del subconjunto candidato, nos referimos a este subconjunto explícitamente por la notación $Candidato\_N(xActual)$. Entonces, el procedimiento de búsqueda tabú puede ser expresado según [7] como se muestra en la Figura 3.2.

Figura 3.2 Método de Búsqueda Tabú

Paso 1 (Inicialización)

Comenzar como en Búsqueda por Entorno, y empezar con el expediente de la historia H vacío.

Paso 2 (Elección y finalización)

Determinar $Candidato\_N(xActual)$ como un subconjunto de $N(H, xActual)$. Seleccionar $xSiguiente$ de $Candidato\_N(xActual)$ para minimizar $c(H, x)$ sobre este conjunto ($xSiguiente$ es llamado elemento de evaluación mayor de $Candidato\_N(xActual)$). Terminar mediante un criterio de parada seleccionado.

Paso 3 (Actualización)

Ejecutar la actualización por el Método de Búsqueda en el Entorno, y actualizar el expediente de la historia H.

La historia H de la búsqueda tabú utiliza una memoria que se basa en atributos para registrar los eventos que han ocurrido durante la búsqueda de acuerdo a [9]. Un atributo de un movimiento de $xActual$ a $xSiguiente$, o de un movimiento ensayo de $xActual$ a una solución tentativa $xEnsayo$, puede abarcar cualquier aspecto que cambie como resultado del movimiento. Algunos tipos naturales de atributos de muestran en la Figura 3.3.

Un movimiento simple puede dar lugar a atributos múltiples. Por ejemplo, un movimiento que cambia los valores de dos variables simultáneamente, como lo es una permutación, puede dar lugar a cada uno de los tres atributos (A1), (A2) y (A3), además de otros atributos de la forma indicada.

Figura 3.3 Ejemplos de Atributos de Movimiento

Para un movimiento $xActual$ a $xEnsayo$:

(A1) Cambio de una variable seleccionada x_j de y a z.

(A2) Cambio de una variable seleccionada x_k de z a y.

(A3) El cambio combinado de (A1) y (A2) tomados juntos.

(A4) Cambio de una función $g(xActual)$ a $g(xEnsayo)$ (donde g puede representar una función que ocurre naturalmente en la formulación del problema o una función que es creada estratégicamente).

Los atributos de movimientos almacenados son a menudo usados en búsqueda tabú para imponer restricciones, que evitan que sean elegidos movimientos que invertirían los cambios representados por estos atributos. Cuando se ejecuta un movimiento de $xActual$ a $xSiguiente$ que contiene un atributo e, se mantiene un registro para el atributo inverso que se denota por $\overline{e}$, para prevenir que ocurra un movimiento que contenga algún subconjunto de tales atributos inversos. En la siguiente figura se muestran algunos tipos de restricciones tabú empleadas comúnmente.

Figura 3.4 Ejemplos de Restricciones Tabú

Un movimiento es tabú si:

(R1) x_j cambia de z a y, previamente x_j cambio de y a z.

(R2) x_k cambia de y a z, previamente x_k cambio de z a y.

(R3) Ocurre al menos una de las restricciones (R1) y (R2).

(R4) Ocurren (R1) y (R2). Es una condición menos restrictiva que la condición anterior.

Una restricción tabú se activa típicamente sólo en el caso en el que sus atributos hayan ocurrido dentro de un rango determinado de iteraciones anteriores a la iteración presente o hayan ocurrido con una cierta frecuencia sobre un periodo de iteraciones más largo. Entonces dichas restricciones como se comenta en [9], dependen de la memoria basada en lo reciente y frecuente de la *BT*.

Se define un atributo como tabú activo cuando su atributo inverso asociado ha ocurrido dentro de un intervalo estipulado de lo reciente o de lo frecuente en movimientos pasados. Un atributo que no es tabú activo se llama tabú inactivo. La condición de ser tabú activo o tabú inactivo se llama el estado tabú de un atributo.

Para mantener el estado de los atributos del movimiento que componen restricciones tabú, y para determinar cuándo son aplicables estás restricciones, se definen funciones básicas de memoria. Las funciones de memoria basada en lo reciente se especifican mediante los valores $ComienzoTabu(e)$ y $FinTabu(e)$, estos vectores identifican, respectivamente, las iteraciones de comienzo y finalización del período tabú para el atributo e, acotando así el período durante el cual e es tabú activo.

En la iteración ite, si e es un atributo del movimiento actual, se define un estado tabú para evitar inversiones. Entonces $ComienzoTabu(e) = ite + 1$, indicando que el atributo inverso $\overline{e}$ mantendrá este estado a lo largo de su período tabú, que denotamos por t. Esto produce $FinTabu(e) = ite + t$, tal que el período para $\overline{e}$ se extiende sobre las t iteraciones de $ite + 1$ a $ite + t$.

Como resultado se puede comprobar si un atributo arbitrario es activo controlando sí $FinTabu(e) \geq IteraciónActual$. Inicializando $FinTabu(e) = 0$ para todos los atributos se asegura de que $FinTabu(e) < IteraciónActual$, y por lo tanto que el atributo e sea tabú inactivo, hasta que se realice la actualización especificada previamente. En este trabajo se utilizan las funciones mencionadas en el diseño del algoritmo.

Además de la memoria basada en lo reciente la *BT* según se dice en [7] utilizan también la memoria basada en lo frecuente. La memoria basada en la frecuencia proporciona un tipo de información que complementa la información proporcionada por la memoria basada en lo reciente, ampliando la base para seleccionar movimientos preferidos.

Las medidas de frecuencia se pueden concebir como proporciones cuyos numeradores representan el número de ocurrencias de un evento particular y cuyos denominadores generalmente representan uno de cuatro tipos de cantidades: (1) el número total de ocurrencias de todos los eventos representados por los numeradores, (2) La suma de los numeradores, (3) el máximo valor del numerador, y (4) la media del valor del numerador. Los denominadores (3) y (4) se expresan como valores absolutos y el denominador (2) se expresa como una suma de valores absolutos.

Figura 3.5 Ejemplos de Medidas de Frecuencia

(Numeradores)

(F1) $\#S\left(x_j = a\right)$.

(F2) $\#S\left(x_j = a \text{ para algún } x_j\right)$.

(F3) $\#S\left(\text{a } x_j = a\right)$.

(F4) $\#S\left(x_j \ cambia\right)$, es decir, $\#S\left(x_j \neq a \text{ a } x_j = a\right)$.

(F5) $\sum_{x \in S\left(x_j = p\right)} c\left(x\right) / \#S\left(x_j = a\right)$.

(F6) Reemplazar $S\left(x_j = a\right)$ en (F5) con $S\left(x_j \neq a \text{ a } x_j = a\right)$.

(F6) Reemplazar $c\left(x\right)$ en (F6) con una medida de influencia $S\left(x_j \neq a \text{ a } x_j = a\right)$.

En un movimiento determinado, los atributos $DesdeAtributo$ están asociados según [7] con cambiar el estado a de un elemento x_j, y los atributos $HaciaAtributo$ están asociados con cambiar el estado b de un elemento x_j. Se denota por $x(1), x(2), ..., x\left(IteraciónActual\right)$ la secuencia de soluciones generadas en el momento presente del proceso de búsqueda, y se denota por S una subsecuencia de esta secuencia de soluciones. La subsecuencia S se puede tratar como un conjunto. Los elementos de S no son necesariamente elementos consecutivos de la secuencia de la solución completa.

Se denota por $S\left(x_j = a\right)$ el conjunto de soluciones en S para las cuales $x_j = a$, y se denota por $\#S\left(x_j = a\right)$ el número de veces que x_j recibe el valor a sobre $x \in S$. Análogamente $S\left(x_j = a \text{ a } x_j = b\right)$ es el conjunto de soluciones en S que resultan por un movimiento que cambia $x_j = a$ a $x_j = b$. Finalmente se denota por $S\left(\text{de } x_j = a\right)$ y $S\left(\text{a } x_j = b\right)$ los conjuntos de soluciones en S que contienen respectivamente $x_j = a$ como un $DesdeAtributo$ o $x_j = b$ como un $HaciaAtributo$. En general, si $AtributoSolucion$ representa cualquier atributo de una solución que puede tomar el papel de un $DesdeAtributo$ o un $HaciaAtributo$ para un movimiento arbitrario denotado por $\left(DesdeAtributo, HaciaAtributo\right)$, entonces:

$$S\left(SolucionAtributo\right) = \left\{x \in S : x \text{ contiene } AtributoSolucion\right\}$$
$$S\left(MovimientoAtributo\right) = \left\{x \in S : x \text{ contiene } MovimientoAtributo\right\}$$
$$S\left(DesdeAtributo\right) = \left\{x \in S : x \text{ contiene } DesdeAtributo\right\}$$
$$S\left(HaciaAtributo\right) = \left\{x \in S : x \text{ contiene } HaciaAtributo\right\}$$

La cantidad $\#S\left(x_j = a\right)$ constituye una medida de residencia, dado que identifica el número de veces que el atributo $x_j = a$ reside en las soluciones de S. Correspondientemente de acuerdo a [7], se define la frecuencia que resulta de dividir tal medida por uno de los denominadores de (1) a (4) como una frecuencia de residencia. Para el numerador $\#S\left(x_j = a\right)$, los denominadores (1) y (2) corresponden ambos a $\#S$, mientras que los denominadores (3) y (4) son dados respectivamente por $Max\left(\#S\left(x_k = b\right) : \text{todo } k, b\right)$ y por $Media\left(\#S\left(x_k = b\right) : \forall k, b\right)$.

Las cantidades $\#S\left(x_j = a \text{ a } x_j = b\right)$, $\#S\left(\text{de } x_j = a\right)$ y $\#S\left(\text{a } x_j = b\right)$ constituyen medidas de transición, dado que identifican el número de veces que x_j cambia de y/o a valores especificados. Asimismo, las frecuencias basadas en tales medidas son llamadas frecuencias de transición. En un movimiento de permutación se tienen dos medidas de transición que ocurren simultáneamente con las cantidades $\#S\left(x_i = a \text{ a } x_j = a\right)$ y $\#S\left(x_j = b \text{ a } x_i = b\right)$, por lo tanto, para obtener la frecuencia de transición solamente se necesita llevar el registro de una de ellas: $f = \#S\left(x_i = a \text{ a } x_j = a\right)$. Los denominadores para crear tales frecuencias incluyen $\#S$, el número total de veces que los cambios indicados ocurren sobre S para diferentes valores $i, j, a \text{ y/o } b$ y cantidades Max y $Media$ asociadas.

En algunas ocasiones las restricciones tabú no son necesariamente válidas debido al uso del criterio de aspiración. Los criterios de aspiración se introducen en la *BT* para determinar cuándo pueden ser reemplazadas las restricciones tabú, eliminando así una clasificación tabú aplicada a un movimiento en otro caso.

El uso apropiado de estos criterios según se dice en [10] puede ser importante para permitir a un método *BT* proporcionar sus mejores niveles de ejecución. La aplicación más elemental solo emplea un tipo simple de criterio de aspiración, que consiste en eliminar una clasificación tabú de un movimiento de ensayo cuando el movimiento conduce a una mejor solución que la mejor obtenida en la historia de la ejecución del algoritmo. Este criterio se sigue usando ampliamente y es el criterio utilizado en el presente trabajo. Puede haber otros criterios de aspiración efectivos para mejorar la búsqueda.

En algunos criterios de aspiración se utiliza el concepto de influencia, que mide el grado de cambio inducido en la estructura de la solución o factibilidad. La influencia a menudo se asocia a la idea de distancias de movimiento, es decir, donde se concibe que un movimiento de mayor distancia tiene mayor influencia. Las estrategias de listas de candidatos implícitamente tienen una influencia diversificante motivando que diferentes partes del espacio de soluciones se examinen en diferentes iteraciones. Con la definición de la *BT* se termina el marco teórico necesario para poder abordar el siguiente capítulo que inicia el diseño del algoritmo que resuelve el *PPHC* a manera de gráfica utilizando esta metaheurística.

4. Un Problema de Horarios Utilizando Gráficas

4.1 Introducción

En este capítulo se introduce un *PPHC* para modelarlo a manera de una gráfica. En las siguientes secciones se muestra el desarrollo del modelo para el ejemplo específico que aqui se presenta y finalmente se define una nomenclatura para definir los problemas con los que se prueba el algoritmo que se diseña en este trabajo.

4.2 Ejemplo de Programación de Clases

En el siguiente ejemplo se desea programar las clases de 15 materias distribuidas en 3 cursos. Cada materia está compuesta de 1,2 ó 3 clases de una hora por semana según se muestra en la figura que sigue:

Figura 4.1 Estructura de los Cursos

Curso	Materias	Hrs/Semana
I	A,B,C K	3 1
II	D, E F,G	3 2
III	H,I,J L,M,N,O	2 1

El problema consiste en asignar 30 clases que conforman las 15 materias, a las 15 horas disponibles por semana (3 horas por día). En el ejemplo no deben existir clases pertenecientes a un mismo curso programadas a una misma hora.

Las clases que pertenecen a una misma materia no deben ser agendadas el mismo día ni en días consecutivos. El número máximo de clases por hora que se permite es igual a 3, ya que de lo contrario se tendrían dos clases pertenecientes a un mismo módulo en el mismo intervalo de tiempo.

Para este ejemplo se utiliza el modelo de gráfica que se introduce a continuación. Se asocia un vértice de la gráfica a cada clase X_i de cada materia del programa $X \in \{A, B, ..., O\}$, con $1 \le i \le n_x$, donde n_x es el número total de clases de la materia. El nodo X_i pertenece al conjunto V_k si la materia X pertenece al curso k, con $k = 1, 2, 3$; El conjunto de 30 vértices es entonces: $V = \{A_1, A_2, A_3; B_1, B_2, B_3; \ldots; N_1; O_1\}$ en tres módulos.

Las 15 horas disponibles en la semana para la programación de las clases están asociadas de manera ordenada a los colores. El color 1 representa la primera hora del lunes, el color 2 la segunda hora del lunes, el color 3 la última hora del lunes y así de la misma manera hasta el color 15 que representa la tercera hora del viernes. De esta manera se identifica el parámetro $c = 15$ como el número válido de colores.

El conjunto de aristas identifica restricciones fuertes entre las clases, de tal manera que una *c*-coloración que representa una programación factible, no puede asignar la misma hora a clases que no son compatibles. Una arista en el ejemplo es la línea que une las clases del mismo curso, evitando colorear los vértices con el mismo color, que es equivalente a programar las clases en el mismo día y hora.

Las aristas complementarias identifican restricciones débiles entre las clases, de tal manera que una *c*-coloración que representa una programación factible, no puede asignar dos o más clases de una materia en el mismo día ni en el día consecutivo. Una arista complementaria en el ejemplo es la línea que une las clases de una misma materia, evitando colorear los vértices con colores muy próximos, que es equivalente a programar las clases el mismo día o en el día consecutivo.

4.3 Modelación de la Gráfica

Se define una gráfica $G = (V, A)$, donde V es el conjunto de vértices (clases) y A es el conjunto de aristas (restricciones fuertes); una coloración (programación) de G es una asignación $C : V \to \{1, 2, \ldots, c\}$, donde $C(i)$ es el color (hora) de $i \in V$, de tal manera que si $\{i, j\} \in A$, entonces $C(i) \neq C(j)$.

Si se toma en cuenta el conjunto de ejes complementarios que se incorpora en los problemas de coloración robusta, la gráfica se relaja eliminando las aristas que unen las clases de una misma materia y manteniendo las aristas en clases que pertenecen a diferentes materias dentro de un mismo curso, reduciendo drásticamente el número de restricciones.

Entonces, el conjunto de aristas del problema es:

$$A = \left\{ \{X_i, Y_j\} \, \forall i, j \,\middle|\, X \neq Y; X, Y \in V_k \text{ para alguna } k = 1, 2, 3 \right\}$$

Entonces las aristas complementarias en la gráfica se agregan para representar la incompatibilidad de las clases que pertenecen a una misma materia. Si se utiliza el nivel de rigidez se pueden penalizar fuertemente las aristas en la gráfica complementaria para hacer que las clases que pertenecen a una determinada materia se programen de acuerdo a las restricciones del programa.

Yáñez y Ramírez (2005) [13] mencionan en su trabajo que el *PCR* consiste en encontrar una k-coloración C de G que minimice $R(C)$ que equivale a encontrar una programación de horarios que minimice los conflictos entre las clases. Para el problema de coloración es necesario entonces establecer cuál es el conjunto de aristas complementarias $\overline{A}$, formado por dos tipos de aristas:

- $\{X_i, X_j\}$, $i \neq j$, para cada materia X. Estas aristas son fuertemente penalizadas para evitar en los extremos de la gráfica la misma coloración. Esta penalización fuerza a evitar la incompatibilidad existente sobre las clases de la misma materia.

- $\{X_i, Y_j\} \forall i, j$, $\forall X \in V_s$, $Y \in V_t$, $X \neq Y$, $s \neq t$. Estas aristas tienen una penalización mínima. Con esta penalización se asegura una distribución uniforme de las clases en las horas disponibles en la universidad y se evita programar todas las clases compatibles al mismo tiempo.

De acuerdo a [12,15] con este enfoque, un problema como el de coloración robusta no puede ser resuelto porque el nivel de rigidez de una coloración se incrementa únicamente con cada arista complementaria cuyos vértices comparten un mismo color. El nivel de rigidez debe aumentar también cuando los colores son suficientemente cercanos, indicando que las clases correspondientes han sido programadas muy próximas en términos de tiempo.

El *PCRG* permite introducir diferentes distancias entre colores que equivalen a la separación de los eventos en el tiempo en el *PPHC*. Utilizando la matriz M es posible identificar situaciones más generales que aquellas consideradas por una *d-distancia* definida para el conjunto de colores y un límite o umbral $\hat{d}$, entonces,

Una forma adecuada para la matriz M del ejemplo es la siguiente:

$$M = \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

La gráfica del problema, con 30 vértices, asociados a las clases en diferentes materias, tiene 3 componentes una por cada curso o módulo. La Figura 2 muestra una gráfica con ejemplos de las aristas principales y complementarias, que representan las incompatibilidades de las clases. Se puede observar el *PCRG* aplicado al ejemplo que aquí se expone.

Las aristas complementarias se penalizan de diferente manera de acuerdo a su origen:

- Aristas complementarias que unen clases de la misma materia son penalizadas con un número suficientemente grande; por ejemplo, $p_{X_iX_j} = 100$ para cada par de clases X_iX_j de la misma materia.

- Aristas complementarias que unen clases de materias en diferentes módulos son penalizadas con una mucha menor cantidad que la anterior; por ejemplo, $p_{X_iY_j} = 1$ para cualquier par de clases X_i, Y_j $\forall X \neq Y$.

Algunas aristas de la gráfica correspondiente al ejemplo que presentan son $\{1,4\}, \{11,18\}, \{22,29\}$. Se tienen aristas complementarias con una penalización de 100 a $\{1,2\}, \{1,3\}, \{4,6\}$ y algunas con penalización de 1 a $\{5,11\}, \{5,21\}, \{20,30\}$.El modelo de programación matemática para el ejemplo que se presenta es entonces:

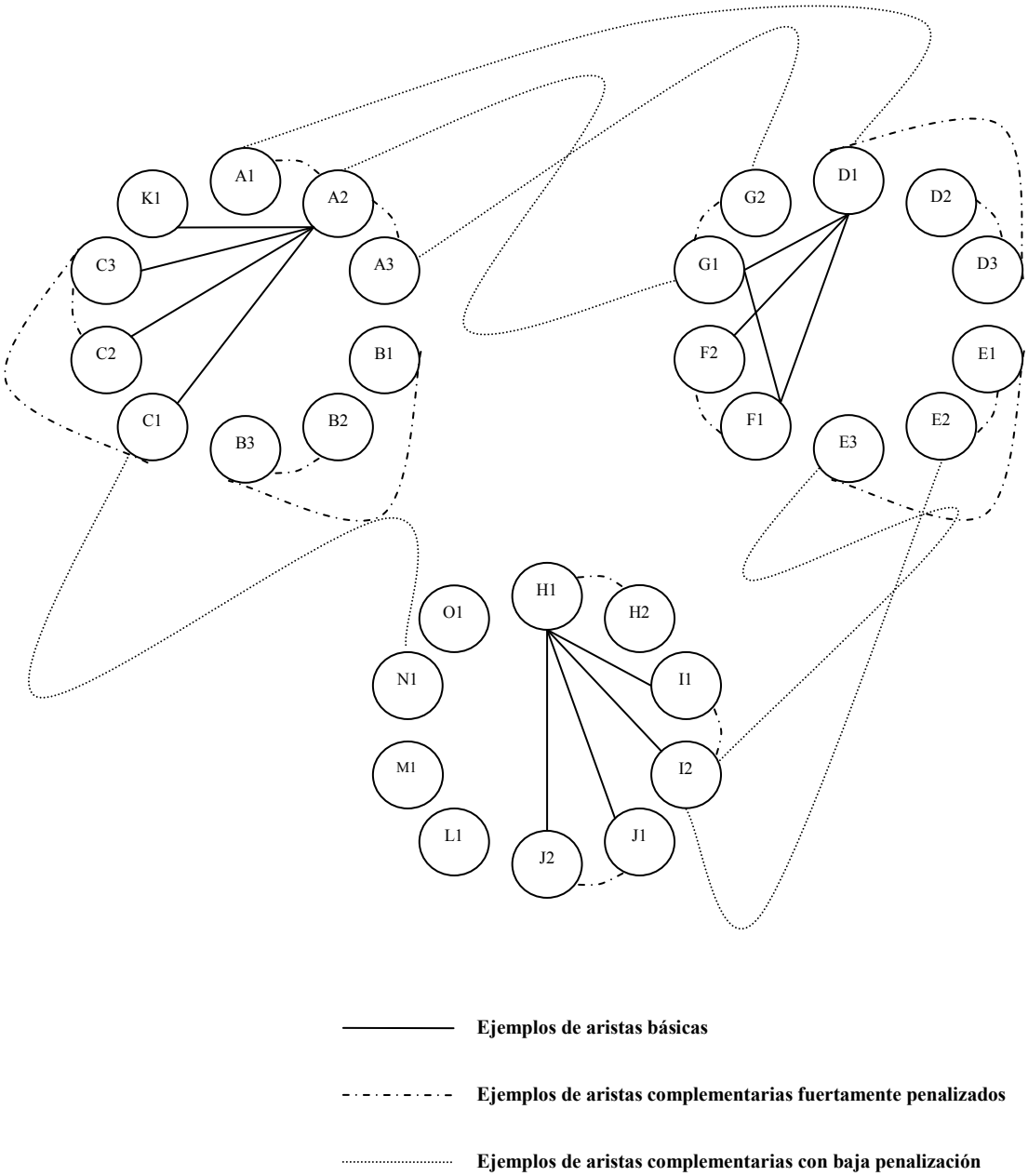
$$Min\ R^{\hat{d}}\left(C^c\right) = 100y_{1,2} + \ldots + y_{5,21} + \ldots + y_{20,30}$$

s.a

$$x_{i,1} + \ldots + x_{i,15} = 1 \quad \forall i$$
$$\cdots$$
$$x_{1,1} + \ldots + x_{4,1} \leq 1$$
$$\cdots$$
$$x_{1,15} + \ldots + x_{4,15} \leq 1$$
$$\cdots$$
$$x_{20,1} + x_{30,1}\left(1 - m_{1,1}\right) + \ldots + x_{30,15}\left(1 - m_{1,15}\right) \leq y_{20,30} + 1$$
$$\cdots$$
$$x_{20,15} + x_{30,1}\left(1 - m_{15,1}\right) + \ldots + x_{30,15}\left(1 - m_{15,15}\right) \leq y_{20,30} + 1$$

El problema que se presenta en este capítulo es solamente uno de los que se utilizan en la experimentación del algoritmo que se desarrolla en el estudio. En la siguiente sección se presenta el grupo de posibles opciones para desarrollar un curso así como su nomenclatura. Juntando varios tipos de cursos de puede desarrollar un programa escolar.

Figura 4.2 Gráfica para el PPHC



4.4 Definición de Problemas

En esta sección se dan los lineamientos para nombrar los diferentes problemas que se utilizan para la experimentación en este trabajo así como en otros que han abordado el mismo problema. Se introduce la nomenclatura desarrollada por Pedro Lara et al. (2010) [15]. Utilizando números y letras se nombran los tipos de cursos existentes, cada curso consta de un grupo de materias; cada materia en total suma 10 clases.

Se determinan 14 estructuras distintas, considerando todas las maneras posibles de generar un curso completo con materias de 1,2 o 3 horas por semana. Los problemas con los que se experimenta se forman al unir varios cursos, el nombre del problema depende de la combinación de cursos dada. La siguiente figura muestra dichas estructuras:

Figura 4.3 Posibles Estructuras para Cursos

Tipo de Curso	No. de Materias con:		
	3 Clases	2 Clases	1 Clase
1	0	0	10
2	0	1	8
3	0	2	6
4	0	3	4
5	0	4	2
6	0	5	0
7	1	0	7
8	1	1	5
9	1	2	3
A	1	3	1
B	2	0	4
C	2	1	2
D	2	2	0
E	3	0	1

Con la información del cuadro anterior se pueden construir problemas con módulos de 10 clases cada uno. El ejemplo que se presenta en este estudio consta de tres cursos. Entonces se debería esperar un nombre para definir el problema que contenga tres letras y/o números.

El primer modulo es de tipo “E” pues contiene tres materias de tres clases cada una y una materia con una sola clase. El segundo modulo es de tipo “D” pues contiene dos materias de tres clases cada una y dos materias consistentes de dos clases. El último modulo es de tipo “4” pues tiene tres materias de dos clases cada una y cuatro materias de una sola clase.

Entonces podemos nombrar el problema enunciado en este capítulo como “ED4”. El orden de los cursos no afecta la estructura del problema por tanto el problema es igual a “D4E”, “4DE” o cualquier otro orden que se dé a las letras y/o números. En este trabajo se utilizan los siguientes problemas:

- ED4
- A42
- ECA864
- 976532
- EDDC96441
- DCB875322
- EEDCCBA87644
- EEDCCBA88776644
- EEDDCCBBAA88776644

Los ejemplos que se utilizan para probar el algoritmo son usados también en [15]. En esta tesis se aportan dos problemas adicionales en el estudio. El primer problema adicional plantea la programación de 150 clases, mientras que en el segundo se aumenta el número a 180 clases Se tiene entonces el caso de una programación de 15 módulos con los siguientes tipos de curso:

- Curso de 3 materias con 3 clases/semana y 1 materia con 1 clase/semana. Este tipo de curso se representa con la letra E.
- Módulo de 2 materias con 3 clases/semana y 2 materias con 2 clases/semana. Dicho curso se representa con la D.
- Curso con 2 materias con 3 clases/semana, 1 materia con 2 clases/semana y 2 materias con 1 clase/semana. Este curso se representa con la C.
- Módulo de 2 materias con 3 clases/semana y 4 materias con 1 clase/semana. Dicho curso se representa con la B.
- Curso de 1 materia con 3 clases/semana, 3 materias con 2 clases/semana y 1 materia con 1 clase/semana. Se representa con la letra A.
- Grupo con 1 materia con 3 clases/semana, 1 materia con 2 clases/semana y 5 materias con 1 clase/semana. Se representa con el número 8.
- Curso de 1 materia con 3 clases/semana y 7 materias con 1 clase/semana. Se representa con el número 7.
- Grupo con 5 materias con 2 clases/semana. Se representa con el número 6.
- Módulo de 3 materias con 2 clases/semana y 4 materias con 1 clase/semana. Dicho curso se representa con el número 4.

El problema de 150 clases contiene 2 cursos del primer tipo, 1 del segundo tipo, 2 del tercero, 1 del cuarto, 1 del quinto y 2 cursos para el sexto, séptimo, octavo y noveno tipo. El problema entonces se define con la nomenclatura EEDCCBA88776644. En el problema de 180 clases se tienen 2 cursos por cada tipo de curso y se define como EEDDCCBBAA88776644.

Los resultados de dichos problemas se muestran en el último capítulo del trabajo, los resultados del *GRASP* se muestran en la última parte del capítulo.

4.5 Algoritmos Heurísticos

En esta sección se expone una breve reseña de algunos métodos desarrollados para la programación de horarios de clases que utilizan la modelización del *PCRG* y utilizan alguna heurística en su resolución. Por ejemplo en [15] utilizan un método *GRASP*, mientras que en [2] se hace uso de algoritmos genéticos.

4.5.1 GRASP

El algoritmo *GRASP* busca minimizar la función $R^{\hat{d}}(C)$ definida como:

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

Donde $\sum_{\{i,j\}\in\bar{A}} p_{ij} y_{ij}$ es el valor de rigidez para la coloración C mientras que $I(c)$ es el número de aristas con una coloración incompatible con la matriz de distancias. El parámetro μ es un factor de peso para cada elemento considerado; según [15] los mejores resultados preliminares con que obtuvieron fue con un valor de $\mu = 5$.

Figura 4.4 Algoritmo GRASP para PPHC

INICIO
Fase 1: Algoritmo Voraz

1. Ordenar las materias de mayor a menor por su número de clases.
2. Colocar las clases en el horario buscando factibilidad. Si la factibilidad no puede ser lograda completamente, un color es asignado a un vértice aleatoriamente.
3. Continuar hasta que todos los vértices en la gráfica sean coloreados,

Fase 2 Método de Mejora

4. Calcular la función $R^{\hat{d}}(C)$ para el problema de coloración formulado.
5. Un cambio de color es hecho aleatoriamente en un vértice aleatorio obteniendo C'.
6. Calcular $R^{\hat{d}}(C')$.
7. Si una mejora es obtenida se cambia a la nueva coloración; SI NO se regresa al paso 2,5.

Se repite la Fase 2 HASTA que no hay mejores coloraciones encontradas.
FIN

El algoritmo *GRASP*, según los resultados encontrados en [15], resulta ser una buena heurística de solución al *PPHC* una vez que se modela como *PCRG*. De acuerdo a los problemas planteados en el estudio se obtuvieron los resultados mostrados en la figura 4.5.

Los resultados obtenidos abarcan ejemplos que van desde 30 clases hasta 120, en los cuales, hubo dos instancias en las que no se llegó a lo que se creía era la rigidez mínima. En el primer caso se demuestra en [2] que este resultado es la solución óptima. Si bien el algoritmo *GRASP* presenta un buen desempeño, otra heurística se podría utilizar en la búsqueda de soluciones al problema.

Figura 4.5 Resultados del GRASP

No. de vértices (clases)	Ejemplo	No. de colores (Intervalos de tiempo)	Clases "fuera de lugar"
30	ED4	15	1
30	A42	15	0
60	ECA864	20	0
60	976532	20	0
90	EDDC96441	30	1
90	DCB875322	30	0
120	EEDCCBA87644	30	0

4.5.2 Algoritmo Genético con Búsqueda Local

El algoritmo genético (*AG*) con búsqueda local (*BL*) tiene como objetivo minimizar la función $R^{\bar{d}}(C)$, que de acuerdo a [2], se define como:

$$Min\ R(C) = \sum_{\{i,j\}\in\overline{A},C(i)=C(j)} p_{ij}$$

$$\text{s.a}\quad C(i) \neq C(j) \quad \forall \{i,j\} \in A$$

La función objetivo se utiliza como función de aptitud para evaluar que tan buena es una solución de acuerdo a su grado de rigidez generalizada. En la Figura 4.6 se muestran los pasos que sigue el algoritmo genético híbrido en la búsqueda de la solución. El *AG* comienza generando una población inicial de manera aleatoria utilizando una tabla con restricciones que evitan colocar clases pertenecientes al mismo curso en una misma hora (formar conjuntos factibles). La solución inicial se evalúa con la función objetivo que se mostró.

Después de evaluar la población inicial se produce una nueva generación utilizando operadores genéticos de cruce que en [2] se definen en dos fases, los operadores genéticos son parte primordial del diseño del algoritmo.

En la primera fase se forman parte de los conjuntos de una manera determinada mientras que en la fase dos el resto se forma de manera aleatoria.

Figura 4.6. AG con Búsqueda Local para PPHC

INICIO
1. Generar una población inicial.
2. Evaluar los elementos de la población inicial.
3. Producir una nueva generación.
4. Realizar la búsqueda local.
5. Actualizar la siguiente generación.

Se repiten los pasos 3 a 5 HASTA cumplir alguna condición establecida
FIN

Con la generación de la nueva población se realiza una búsqueda local intercambiando el color entre conjuntos. Se evalúa el movimiento realizado y se actualiza la solución. Se repite el procedimiento hasta que no haya una mejora posible. De acuerdo a las evaluaciones se actualiza la siguiente generación siguiendo los lineamientos establecidos.

Los resultados obtenidos abarcan ejemplos que van desde 30 clases hasta 120, en los cuales, hubo únicamente una instancia en la que existe una clase fuera de lugar. Entonces se tiene una hora con tres clases programadas.

Figura 4.7 Resultados del AG Híbrido

No. de vértices (clases)	Ejemplo	No. de colores (Intervalos de tiempo)	Clases “fuera de lugar”
30	ED4	15	1
30	A42	15	0
60	ECA864	20	0
60	976532	20	0
90	EDDC96441	30	0
90	DCB875322	30	0
120	EEDCCBA87644	30	0

En [2] se menciona que en el problema ED4, la solución óptima es entonces la que tiene una clase fuera de lugar, y por tanto, no todas las horas disponibles tienen el mismo número de clases. Utilizando el principio del palomar o principio de Dirichlet se realiza una demostración por contraejemplo.

En el siguiente capítulo, se presenta un algoritmo que se desarrolló como parte de este trabajo de tesis, el cual utiliza la heurística de Búsqueda Tabú para encontrar buenas soluciones al *PPHC* que se modela a manera de gráfica. El algoritmo busca la gráfica con el menor nivel de rigidez de acuerdo a las restricciones de un problema.

El algoritmo se prueba con 9 ejemplos y los resultados se comparan con los resultados obtenidos por la heurística del *GRASP* y por el *AG* híbrido, que se presentaron en este capítulo.

5. La Busqueda Tabú para un Problema de Horarios

5.1 Introducción

En este capítulo se desarrolla toda la parte del diseño y experimentación del algoritmo que se propone para resolver el *PPHC* utilizando la modelación del *PCRG*. Este capítulo muestra a detalle los pasos a seguir para poder generar soluciones factibles de programación de horarios utilizando BT. En la siguiente sección se introduce el concepto de matriz de locaciones que hace posible el uso de la memoria tabú.

5.2 Matriz de Locaciones

El diseño del algoritmo de BT para el *PPHC* con el *PCRG* comienza introduciendo la matriz de locaciones para poder realizar permutaciones y almacenarlas en la memoria de la BT. En una matriz $L_{r\times s}$ existen n elementos llamados locaciones en donde, r representa el número total de colores disponibles y s el número de cursos en el problema. Cada locación $l_{x,y}$, en donde $x \in \{1,2,...,r\}$ e $y \in \{1,2,...,s\}$, representa un lugar en la matriz de locaciones, el cual puede estar vacío o puede contener a una sola clase X_i del programa. La matriz de locaciones es entonces:

$$L = \begin{pmatrix} l_{11} & \cdots & l_{1s} \\ \vdots & \ddots & \vdots \\ l_{r1} & \cdots & l_{rs} \end{pmatrix}$$

Las locaciones existentes en cada fila de la matriz no pueden exceder el número de módulos s dada la restricción de que no puede haber dos clases del mismo curso programadas a la misma hora. Para el problema ilustrativo del primer capítulo (ED4) se tiene una tabla como la que se muestra en la figura 5.1.

La tabla es la estructura que almacena las clases del programa, cada registro de la tabla corresponde a una locación de la matriz $L_{r\times s}$, el cual puede estar vacío (cero para realizar los cálculos) o puede contener a una sola clase del problema.

Para generar una solución inicial las clases pueden ser acomodadas en orden o aleatoriamente en los registros de la tabla. En la figura 5.2 se puede observar una solución inicial generada ordenadamente, en total la figura muestra 45 locaciones de las cuales 15 están vacías y el resto están ocupadas por las clases del programa. Esta representación permite utilizar las permutaciones como operadores para generar soluciones de una manera sencilla.

Figura 5.1 Locaciones para el Problema ED4

Día	*Colores/Cursos*	*1*	*2*	*3*
Lunes	*1*	$l_{1,1}$	$l_{1,2}$	$l_{1,3}$
	2	$l_{2,1}$	$l_{2,2}$	$l_{2,3}$
	3	$l_{3,1}$	$l_{3,2}$	$l_{3,3}$
Martes	*4*	$l_{4,1}$	$l_{4,2}$	$l_{4,3}$
	5	$l_{5,1}$	$l_{5,2}$	$l_{5,3}$
	6	$l_{6,1}$	$l_{6,2}$	$l_{6,3}$
Miércoles	*7*	$l_{7,1}$	$l_{7,2}$	$l_{7,3}$
	8	$l_{8,1}$	$l_{8,2}$	$l_{8,3}$
	9	$l_{9,1}$	$l_{9,2}$	$l_{9,3}$
Jueves	*10*	$l_{10,1}$	$l_{10,2}$	$l_{10,3}$
	11	$l_{11,1}$	$l_{11,2}$	$l_{11,3}$
	12	$l_{12,1}$	$l_{12,2}$	$l_{12,3}$
Viernes	*13*	$l_{13,1}$	$l_{13,2}$	$l_{13,3}$
	14	$l_{14,1}$	$l_{14,2}$	$l_{14,3}$
	15	$l_{15,1}$	$l_{15,2}$	$l_{15,3}$

Figura 5.2 Solución Inicial ED4

Día	*Colores/Cursos*	*1*	*2*	*3*
Lunes	*1*			
	2			
	3			
Martes	*4*			
	5			
	6	A_1	D_1	H_1
Miércoles	*7*	A_2	D_2	H_2
	8	A_3	D_3	I_1
	9	B_1	E_1	I_2
Jueves	*10*	B_2	E_2	J_1
	11	B_3	E_3	J_2
	12	C_1	F_1	L_1
Viernes	*13*	C_2	F_2	M_1
	14	C_3	G_1	N_1
	15	K_1	G_2	O_1

Una permutación en la matriz se define cuando $l_{x,y} = X_i$ y $l_{w,z} = Y_j$ cambian sus valores entre sí para dar lugar a una nueva estructura $l_{x,y} = Y_j$ y $l_{w,z} = X_i$. El movimiento puede modificar la estructura de la tabla mediante el intercambio del contenido de un par de registros en una misma columna. Un ejemplo en la tabla anterior es $l_{7,1} = A_2$ y $l_{4,1} = 0$ por $l_{7,1} = 0$ y $l_{4,1} = A_2$. Las figuras 5.3 y 5.4 muestran este movimiento de permutación en el arreglo de locaciones.

En la búsqueda tabú es primordial el proceso de la memoria adaptativa para llevar el registro de las operaciones realizadas y dirigir una búsqueda inteligente de una solución que satisfaga adecuadamente los requerimientos del problema. Para simplificar la estructura y representarla en una tabla, se le asigna una sola posición o ubicación a cada locación de la solución.

La posición de una clase en la solución puede ser determinada utilizando la transformación $u_i = r(y-1)+x$, en donde $i \in \{1,2,...,n\}$ y $n = r \times s$. Una permutación entonces, en términos de posiciones, se define cuando $u_x = X_i$ y $u_y = Y_j$ cambian sus valores entre sí para dar lugar a una nueva estructura $u_x = Y_j$ y $u_y = X_i$. Un ejemplo de una permutación con posiciones en la tabla del ejemplo es $u_7 = A_2$ y $u_4 = 0$ por $u_7 = 0$ y $u_4 = A_2$.

Figura 5.3 Estructura antes de un Movimiento de Permutación

Día	*Colores/Cursos*	*1*	*2*	*3*
Lunes	*1*			
	2			
	3			
Martes	*4*			
	5			
	6	A_1	D_1	H_1
Miércoles	*7*	A_2	D_2	H_2
	8	A_3	D_3	I_1
	9	B_1	E_1	I_2
Jueves	*10*	B_2	E_2	J_1
	11	B_3	E_3	J_2
	12	C_1	F_1	L_1
Viernes	*13*	C_2	F_2	M_1
	14	C_3	G_1	N_1
	15	K_1	G_2	O_1

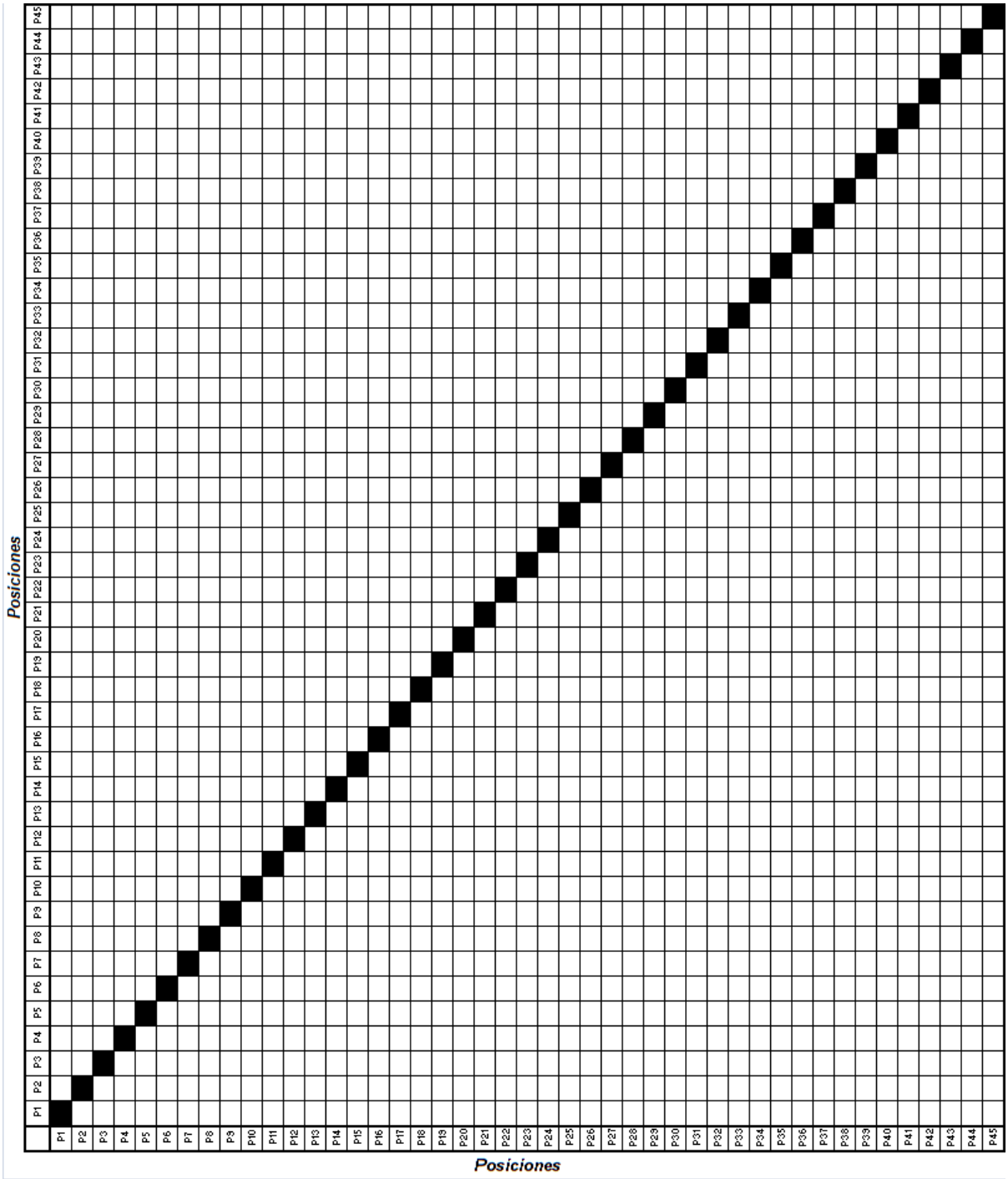
Figura 5.4 Estructura después de un Movimiento de Permutación

Día	*Colores/Cursos*	*1*	*2*	*3*
Lunes	*1*			
	2			
	3			
Martes	*4*	A_2		
	5			
	6	A_1	D_1	H_1
Miércoles	*7*		D_2	H_2
	8	A_3	D_3	I_1
	9	B_1	E_1	I_2
Jueves	*10*	B_2	E_2	J_1
	11	B_3	E_3	J_2
	12	C_1	F_1	L_1
Viernes	*13*	C_2	F_2	M_1
	14	C_3	G_1	N_1
	15	K_1	G_2	O_1

Figura 5.5 Posiciones en ED4

Día	*colores/cursos*	*1*	*2*	*3*
Lunes	*1*	u_1	u_{16}	u_{31}
	2	u_2	u_{17}	u_{32}
	3	u_3	u_{18}	u_{33}
Martes	*4*	u_4	u_{19}	u_{34}
	5	u_5	u_{20}	u_{35}
	6	u_6	u_{21}	u_{36}
Miércoles	*7*	u_7	u_{22}	u_{37}
	8	u_8	u_{23}	u_{38}
	9	u_9	u_{24}	u_{39}
Jueves	*10*	u_{10}	u_{25}	u_{40}
	11	u_{11}	u_{26}	u_{41}
	12	u_{12}	u_{27}	u_{42}
Viernes	*13*	u_{13}	u_{28}	u_{43}
	14	u_{14}	u_{29}	u_{44}
	15	u_{15}	u_{30}	u_{45}

Figura 5.6 Memoria Tabú para ED4



Los intercambios de registros ocurren en la misma columna para prevenir colocar dos clases del mismo curso a una misma hora. Estos intercambios dan lugar únicamente a varias combinaciones factibles de clases en un determinado intervalo de tiempo.

Las posiciones permiten identificar de una manera sencilla los intercambios que se hacen durante la ejecución del algoritmo y llevar un registro en una tabla como parte de la memoria adaptativa de la heurística.

En el ejemplo ED4 se necesita una tabla con 45 posiciones que permitan colocar la información de la búsqueda. La parte de la tabla que se ubica en la esquina superior derecha, por encima de la diagonal de la misma, se utiliza para contener la información relacionada con que tan reciente fue un movimiento dentro del proceso de búsqueda, mientras que la parte inferior izquierda que se encuentra por debajo de la diagonal, almacena la frecuencia con que dichos movimientos han ocurrido.

Para ejemplificar como se registran los eventos en la memoria tabú del algoritmo propuesto, se toma como solución inicial la mostrada en la Figura 5.2 para el problema ED4. Después se suponen tres mejores movimientos de permutación, $u_7 = A_2$ con $u_4 = 0$, $u_{29} = G_1$ con $u_{17} = 0$ y $u_{22} = D_2$ con $u_{23} = D_3$ en la primera, segunda y tercera iteración respectivamente.

Para calcular $FinTabu$ en la memoria basada en lo reciente, se toma un valor de $t = 10$, Para registrar las medidas de transición en la memoria de frecuencia se utiliza $f_x = \#S\left(u_x = X_i \text{ a } u_y = X_i\right)$ y cada que se presenta una cierta transición se incrementa se registro. La tabla 5.8 muestra el estado de la memoria tabú después de las primeras 3 iteraciones. Los movimientos y su registro en una memoria dinámica son los elementos básicos del algoritmo.

5.3 El modelo heurístico

El objetivo del algoritmo heurístico es minimizar la función objetivo $R^d\left(C\right)$ que se define como:

$$Min\ R^d\left(C\right) = \sum_{\{i,j\} \notin A} p_{ij}$$

Donde:

$$p_{ij} = \begin{cases} 100 & m_{C(i),C(j)} = 1 \quad \{i,j\} \in \overline{A} \\ 1 & m_{C(i)C(j)} = 0 \quad \{i,j\} \in \overline{A} \end{cases}$$

La función objetivo evalúa cada distancia, de acuerdo a la matriz M, entre los elementos que ocupan la matriz de locaciones (clases), asignado las penalizaciones por rigidez correspondientes. Para empezar el algoritmo es necesario inicializarlo con una solución que puede ser una como la mostrada en la Figura 5.2 para ED4.

Figura 5.7 Búsqueda Tabú para PCRG

INICIO

1. Generar una solución inicial.
2. Calcular la rigidez inicial.
3. Crear una lista de candidatos con permutaciones aleatorias.
4. Seleccionar el mejor candidato.
5. Actualizar los valores del sistema.
6. Repetir pasos 3-5 HASTA que $\%\Delta$ sea menor igual que $\%E$ o hasta realizar ϕ iteraciones.
7. Generar los resultados.

FIN

Una vez que se genera una solución inicial, ya sea de manera aleatoria o con un cierto orden, se procede a evaluarla con la función de rigidez descrita anteriormente, obteniendo una rigidez inicial $R_0^{\,d}(C)$. La evaluación se convierte en el valor $MejorRigidez$ en la iteración inicial $ite = 0$. Este valor será reemplazado por $R^d(C)$ cuándo $R^d(C) < MejorRigidez$. Cuando un movimiento tenga una restricción tabú reciente, esta podrá invalidarse si se cumple la condición anterior.

El siguiente paso es crear una lista de μ candidatos que nos de variantes en la dirección de la búsqueda de la solución sin tener que explorar el espacio completo de soluciones. Los pasos para la generación de un candidato se muestran en la figura a continuación:

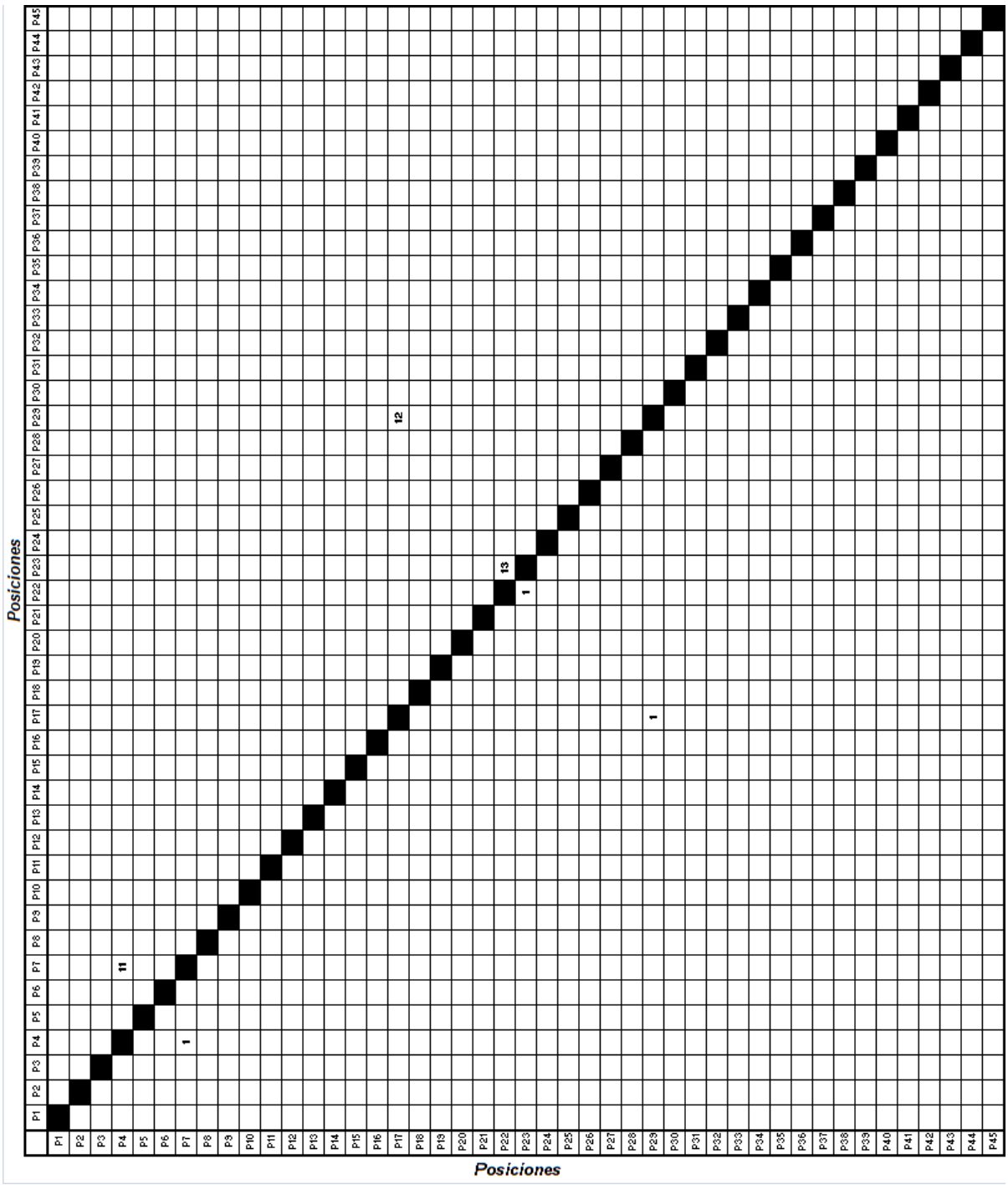
Figura 5.8 Crear una Lista de Candidatos

INICIO

1. Escoger un módulo s_x aleatoriamente.
2. Seleccionar dos colores r_{y1} y r_{y2} al azar.
3. Obtener la posiciones de los colores seleccionados u_i y u_j.
4. Realizar el intercambio de clases entre las posiciones.
5. Calcular la rigidez $R^d(C)$ con el cambio incluido.
6. Evaluar la rigidez penalizada $R^d(C)^*$.
7. Obtener restricción de lo reciente $FinTabu(e)$ para u_i y u_j.
8. Repetir los pasos 1-7 HASTA μ veces.

FIN

Figura 5.9 Memoria Tabú para ED4 después de la Tercera Iteración



En la Figura 5.10 se muestra un ejemplo de la lista de candidatos para el problema ED4. La figura muestra los parámetros que se mencionan en la Figura 5.8 para la iteración número 1000 en uno de los ejercicios realizados considerando $t = 5$.

Figura 5.10 Ejemplo de Lista de Candidatos para ED4

Índex	Modulo	Color1	Color2	Pos1	Pos2	R	R*	FinTabu
1	2	5	8	20	23	315	315	0
2	3	8	15	38	45	115	115	219
3	2	1	13	16	28	216	216	168
4	2	3	3	18	18	115	229	989
5	3	11	14	41	44	116	116	486
6	2	5	13	20	28	216	216	355
7	3	8	14	38	44	116	116	558
8	2	4	8	19	23	216	216	563
9	2	5	7	20	22	216	216	0
10	2	2	14	17	29	115	116	909

La función de rigidez penalizada permite integrar la memoria de frecuencia en el algoritmo. El algoritmo heurístico utiliza esta información para expandir la búsqueda a otros lugares del espacio de soluciones cuando una zona ha sido muy explorada, aumentando su frecuencia e incrementando su rigidez penalizada:

$$R^d(C)^* = R^d(C) + \rho\left(\%f, R^d(C)\right)$$

Donde:

$$\rho\left(\%f, R^d(C)\right) = \begin{cases} \left\lceil \frac{f}{ite}\left(R^d(C)\right) \right\rceil & Mod(num, ite)/ite \geq 1/2 \\ \left\lfloor \frac{f}{ite}\left(R^d(C)\right) \right\rfloor & Mod(num, ite)/ite < 1/2 \end{cases} \quad \text{y}$$

$$Mod(num, ite) = num - ite\left(\left\lfloor \frac{num}{ite} \right\rfloor\right), \quad num = f \times R^d(c)$$

La rigidez y la rigidez penalizada son parámetros que permiten la selección de un candidato de la lista generada por el algoritmo. Podría ser que el último intercambio de la Figura 5.10 $u_{17} = 2$ y $u_{29} = 14$, exista una frecuencia $f = 5$ de dicho intercambio en la iteración en curso $ite = 1000$, lo cual aumenta su rigidez de $R^d(C) = 115$ a una rigidez penalizada $R^d(C)^* = 116$. En la Figura 5.11 se pueden observar los pasos para la elección del mejor candidato:

Figura 5.11 Seleccionar el Mejor Candidato

INICIO

1. Ordenar la lista de candidatos de menor a mayor grado de rigidez.
2. Seleccionar el cambio propuesto SI la rigidez $R^d(C) < MejorRigidez$ o SI el número de iteración $\phi > FinTabu$.
3. Repetir el paso número 2 HASTA cumplir alguna de las condiciones anteriores o HASTA haber recorrido los μ candidatos.
4. Continuar el proceso si no ha habido alguna selección o ir al paso número 9 de lo contrario.
5. Ordenar la lista de candidatos de menor a mayor grado de rigidez penalizada.
6. Seleccionar el intercambio SI $\phi > FinTabu$.
7. Repetir el paso número 6 HASTA cumplir con la condición anterior o HASTA haber recorrido los μ candidatos.
8. Seleccionar el movimiento con menor Rigidez Penalizada $R^d(C)^*$ en caso de no haber cumplido con ninguna condición o ir al paso número 9 en caso contrario.
9. Modificar la tabla de solución realizando el intercambio seleccionado.

FIN

Cuando la selección del candidato se ha realizado, el siguiente paso es la actualización de los valores en la memoria adaptativa de la BT. Los pasos 3, 4 y 5 descritos en la Figura 5.7 se repiten cíclicamente hasta que porcentaje de desviación $\%\Delta$, sea menor igual que el porcentaje de error admisible $\%E$. El porcentaje delta $\%\Delta$ se define como:

$$\%\Delta = \frac{\left(R^d(C) - Opt\right)}{Opt}$$

Para el cálculo de este parámetro es necesario también definir un valor óptimo de una programación de horarios de clases. Si se tienen r colores y α clases, entonces, se busca acomodar de manera equitativa a cada una de ellas, buscando el mismo número de clases por hora cph. Si existe un par de materias en una hora determinada y dichas materias pertenecen a diferentes cursos, la única penalización recibida tendrá el valor de 1 de acuerdo a las definiciones de penalización recibidas en las aristas complementarias. Tenemos entonces que:

$$Opt = \sum_{i=0}^{(r-Mod(\alpha,r))} \binom{cph}{2} + \sum_{j=0}^{Mod(\alpha,r)} \binom{cph+1}{2}$$

Donde:

$$cph = \alpha - Mod(\alpha, r)/r\,, \quad Mod(\alpha, r) = \alpha - r\left(\left\lfloor \alpha/r \right\rfloor\right)$$

El $\%\Delta$ obtenido en cada iteración se comparará contra el error admisible $\%E$ en el algoritmo, cuando el porcentaje delta sea menor o igual que el porcentaje de error admisible el algoritmo heurístico detendrá su ejecución. El algoritmo obtuvo deltas menores que los errores admisibles en los ejercicios, logrando la convergencia en todos los problemas en donde se implementó. La siguiente sección presenta los resultados obtenidos.

5.4 Resultados del Algoritmo de Búsqueda Tabú

Los resultados del algoritmo BT se presentan en esta parte del trabajo de investigación proponiendo mejoras al método heurístico para desarrollar una herramienta de mejor desempeño en la resolución del problema de coloración robusta generalizado en futuras investigaciones.

Los resultados de los problemas presentados en el trabajo fueron validados observando que ninguna de las estructuras de horarios arrojadas por el modelo heurístico, tuviera por lo menos una clase que no cumpliera con alguna de las restricciones establecidas.

Se puede observar en la Figura 5.12 los resultados para el problema ED4. En ninguna clase perteneciente a un mismo curso se repite la hora (color). También se puede notar que ninguna de las clases que pertenecen a una misma materia es programada el mismo día ni en días consecutivos. Existe solo una hora (7) que contiene tres clases, que son las máximas permitidas por hora, todas las demás están uniformemente repartidas en el horario disponible de la universidad. En tan solo 3,602 iteraciones se obtuvo un resultado con un error de 6.67%, el resultado se puede considerar como bueno.

La tabla que se muestra a continuación muestra el desempeño del algoritmo de búsqueda tabú para el PCRG en 9 ejemplos tomados para las pruebas con este nuevo modelo heurístico. Dos de los ejemplos aquí presentados son de mayor extensión que cualquiera presentado en el trabajo de P. Lara et al [15]. Los resultados son considerablemente buenos y obtenidos en una cantidad de tiempo razonable.

Figura 5.12 Resultados ED4

Índice	Curso	Asignatura	Clase	Color
1	1	1	1	3
2	1	1	2	15
3	1	1	3	8
4	1	2	1	14
5	1	2	2	7
6	1	2	3	2
7	1	3	1	9
8	1	3	2	13
9	1	3	3	1
10	1	4	1	6
11	2	5	1	15
12	2	5	2	8
13	2	5	3	2
14	2	6	1	14
15	2	6	2	1
16	2	6	3	7
17	2	7	1	4
18	2	7	2	10
19	2	8	1	5
20	2	8	2	12
21	3	9	1	3
22	3	9	2	10
23	3	10	1	11
24	3	10	2	5
25	3	11	1	12
26	3	11	2	4
27	3	12	1	7
28	3	13	1	9
29	3	14	1	13
30	3	15	1	6

Contador	**3602**
Óptimo	**15**
Mejor	**16**
Rigidez	**16**
Error	**6.67%**
Admisible	**7.00%**

Figura 5.13 Resultados Obtenidos vs GRASP

No. de vértices (clases)	Ejemplo	No. de colores (Intervalos de tiempo)	Clases “fuera de lugar” en GRASP	Clases “fuera de lugar” en BT
30	ED4	15	1	1
30	A42	15	0	0
60	ECA864	20	0	0
60	976532	20	0	0
90	EDDC96441	30	1	0
90	DCB875322	30	0	0
120	EEDCCBA87644	30	0	0
150	EEDCCBA88776644	40	0	0
180	EEDDCCBBAA88776644	40	0	0

El algoritmo muestra un buen aprovechamiento de los espacios con ninguna clase fuera de lugar, exceptuando el ejercicio ED4, en donde existe una clase fuera de lugar al igual que en el trabajo de P. Lara et al [15]. La búsqueda tabú para PCRG muestra una mejoría en uno de los casos de estudio sobre los resultados expuestos en el GRASP para PCRG.

Figura 5.14 Resultados Obtenidos vs AG

No. de vértices (clases)	Ejemplo	No. de colores (Intervalos de tiempo)	Clases "fuera de lugar" en AG	Clases "fuera de lugar" en BT
30	ED4	15	1	1
30	A42	15	0	0
60	ECA864	20	0	0
60	976532	20	0	0
90	EDDC96441	30	0	0
90	DCB875322	30	0	0
120	EEDCCBA87644	30	0	0
150	EEDCCBA88776644	40	0	0
180	EEDDCCBBAA88776644	40	0	0

El algoritmo de búsqueda tabú presenta soluciones de la misma calidad que los expuestos por el algoritmo genético híbrido. El resultado obtenido en el problema ED4 es el óptimo de acuerdo a la demostración que C. Pérez [14] realiza en su trabajo. Un estudio de los tiempos que tardan los algoritmos en converger en una buena solución puede ser un buen parámetro comparativo entre los dos métodos.

El tiempo que el algoritmo tarda en obtener una solución adecuada crece exponencialmente conforme el tamaño del problema, entonces para los problemas con un mayor número de vértices el tiempo requerido para la obtención de un resultado aceptable es mayor.

Figura 5.15 Horario de Clase (ED4)

Hora	Lunes	Martes	Miércoles	Jueves	Viernes
1	*C(III)* *E(III)*	*F(II)* *J(II)*	*B(III)* *E(III)* *L(I)*	*F(II)* *H(II)*	*C(III)* *N(I)*
2	*B(III)* *D(III)*	*G(II)* *I(II)*	*A(III)* *D(III)*	*I(II)*	*B(III)* *E(III)*
3	*A(III)* *H(II)*	*K(I)* *O(I)*	*C(III)* *M(I)*	*G(II)* *J(II)*	*A(III)* *D(III)*

Para obtener los resultados mostrados se programó en Visual Basic un complemento ejecutable en Microsoft Excel. El código de la programación se encuentra en los Anexos del trabajo.

El algoritmo para el PCRG se basa en la búsqueda tabú básica, por tanto, se propone la creación de un híbrido de este procedimiento con algún otro que de beneficios adicionales al ya existente. Como ejemplo, se podría utilizar un algoritmo heurístico sencillo para generar la solución inicial en lugar de generarla aleatoriamente o con un cierto orden como en este trabajo. Esto podría ahorrar tiempo al algoritmo para converger en una solución aceptable.

Otra característica que se podría explotar abiertamente en la BT es la dimensión de la influencia, la cual podría basarse en la distancia que existe entre los colores. Mientras mayor sea la distancia entre los colores mayor sería el grado de influencia del intercambio y por el contrario, mientras menor sea la distancia entre los colores menores sería el grado de influencia del movimiento. Cualquiera de estas propuestas podría mejorar el método que aquí se desarrolla, ya sea incorporándolas separadamente o al mismo tiempo.

La búsqueda tabú que se presenta en este trabajo resulta ser eficiente en la exploración de buenas soluciones al problema de planificación de horarios. El algoritmo BT puede generar horarios de clases respetando las restricciones y aprovechando los recursos existentes. Existen áreas de oportunidad en el diseño del procedimiento aquí descrito, que podrían mejorar significativamente su desempeño y que pueden ser futuras líneas de investigación en el tema.

Conclusiones

En este trabajo se presentó un algoritmo para resolver el *PPHC*, el cual se enfocó como un *PCRG*. El problema se representó con una gráfica para facilitar su modelación, así como también, la incorporación de una heurística.

El objetivo de esta tesis fue crear una alternativa de solución al *PPHC* con restricciones de separación de las clases en el tiempo. En este desarrollo se creó un algoritmo de *BT* para resolver el *PPHC* a manera de gráfica, para problemas de considerable extensión, que pueda ser aplicado en una amplia variedad de casos prácticos. El objetivo planteado en la introducción de la tesis fue cubierto.

En el trabajo se definió inicialmente el marco teórico necesario para abordar el diseño y desarrollo del algoritmo. En el Capítulo 1 se definió el *PPHC*, en el Capítulo 2 se introdujeron los problemas de coloración, desde el *PCM* hasta el *PCRG* y en el Capítulo 3 se dieron los conceptos básicos de la *BT*. Despúes se utilizó un ejemplo específico de un problema de horarios para utilizar la estructura de una gráfica y poder modelar el problema para incorporar la metaheurística de la *BT*. En el último capítulo se implementó la *BT* utilizando la Matriz de Locaciones para definir el vecindario de búsqueda y estructurar la memoria tabú del algoritmo, que permitió salir de los óptimos locales y encontrar soluciones lo suficientemente buenas.

El algoritmo encontró soluciones equiparables a los algoritmos mencionados en el Capítulo 4 que abordan la misma problemática. El algoritmo resulta con un mejor desempeño que el algoritmo que utiliza la metaheurística de GRASP [15] encontrando una mejora en uno de los ejemplos mostrados. Todas las soluciones arrojadas por el algoritmo son totalmente factibles y distribuyen de una buena manera las clases que conforman el programa educativo.

El algoritmo de *BT* se limito a probarse para ejemplos que van desde programas conformados por 30 clases hasta programas que contienen 180 clases en su estructura. El algoritmo solamente esta diseñado para incorporar restricciones de separación de las clases en el tiempo, no contempla restricciones de algún otro tipo. En las posibles futuras líneas de investigación se podría intentar incorporar algunos tipos de restricciones que no han sido consideradas por este algoritmo. El algoritmo también está sujeto a mejoras, utilizando la intensificación de la *BT*, tal vez se podría mejorar la aceleración de la convergencia del algoritmo en una buena solución. Muchos de los investigadores en esta línea recomiendan utilizar modelos híbridos para incorporar mejoras en la búsqueda de la solución. Se podría crear un modelo híbrido utilizando este algoritmo con algún otro que haya presentado un buen desempeño.

Referencias

[1] A. Schaerf. (1996). Tabu Search Techniques for Large High-School Timetabling Problems, AAAI, 96 363-368.

[2] C. Pérez, J. Ramírez. Un Algoritmo Genético para un Problema de Horarios con Restricciones Especiales, aceptado en la Revista de Matemática: Teoría y Aplicaciones (2010). Costa Rica.

[3] D. de Werra. (1985). An Introduction to Timetabling, European Journal of Operations Research, 19, 151-162.

[4] D. de Werra. (1996). Some Combinatorial Models for Course Scheduling, European Journal of Operations Research, Vol. 96, 504-513.

[5] E. K. Burke, K. Jackson, J. Kingston. (1997). Automated University Timetabling: The State of the Art, The Computer Journal, Vol. 40 565-571.

[6] E. K. Burke, S. Petrovic. (2002). Recent Research Directions in Automated Timetabling, European Journal of Operational Research, 140 266-280.

[7] F. Glover, B. Melián. (2003). Búsqueda Tabú, Revista Iberoamericana de Inteligencia Artificial. No.19 49-48.

[8] F. Glover, F. Laguna. (1997). Tabu Search, Kluwer Academic Publishers, Norwell, MA, USA.

[9] F. Glover. (1989). Tabu Search-Part I, ORSA Journal on Computing Vol. 1, No. 3, Verano.

[10] F. Glover. (1990). Tabu Search-Part II, ORSA Journal on Computing Vol. 2, No. 1, Invierno.

[11] F. S. Roberts. (2003). T-Colorings of graphs: recent results and open problems, Discrete Mathematics, 93 229-245.

[12] J. Ramírez, Extensiones del Problema de Coloración de Grafos, Tesis Doctoral, Universidad Complutense de Madrid, Madrid, España, (2001). **http://www.ucm.es/BUCM/tesis/mat/ucm-t24770.pdf**

[13] J. Ramírez, J. Yáñez. (2003). The Robust Coloring Problem, European Journal of Operational Research, 148 546-558.

[14] M. W. Carter, G. Laporte. (1998). Recent Developments in Practical Course Timetabling, PATAT'97, LNCS 1408 3-19.

[15] P. Lara, R. López, J. Ramirez, J. Yañez. (2010). A Model for Timetabling Problems with Period Spread Constraints, Journal of the Operation Research Society. 1-6.

[16] R. Lewis. (2008). A Survey of Metaheuristics-based Techniques for University Timetabling Problems, Springer-Verlag, OR Spectrum, 30:167-190.

[17] S. Kanmani, M. Nandhini. (2009). A Survey of Simulated Annealing Methodology for University Course Timetabling, International Journal of Recent Trends in Engineering, Vol.1, No.2.

[18] Z. Bratković, T. Herman, V. Omrč, M. Čupić, D. Jakobović. (2009). University Course Timetabling with Genetic Algorithm: a Laboratory Exercises Case Study, EvoCOP 2009, LNCS 5482, 240-251.

Anexos

A1. Código de Programación de la Forma

```
Private Sub CommandButton1_Click()

GetData
FindOptimum
StartGraph

Cntr = 1

Do While (Error > Dev) And Cntr < (Iterations - t1)
   SwapLocations
   SelectSwap
   UpdateValues
   Cntr = Cntr + 1
Loop

Output

If Cntr = (Iterations - t1) And Error >= Dev Then
   MsgBox ("El programa no pudo converger en una solución factible")
Else
   'Do Nothing
End If

UserForm1.Hide

End Sub

Private Sub CommandButton2_Click()

Unload UserForm1
UserForm1.Hide

End Sub
```

A2. Código de Programación de Módulo

```
'-----------------------------------------------------------------------------------------------------------------------
'Program: Taboo Search for GRCP.XLS
'Function: Find a feasible solution for the GRCP, using the TS Methodology
'Modeler and programmer: Mario Aboytes Ojeda
'Date: 10/08/2009
'-----------------------------------------------------------------------------------------------------------------------

Option Explicit
Dim Aux0(60, 8) As Integer                  'Auxiliar Matrix
Dim Aux1 As Integer                         'Auxiliar Vatriable No1
Dim Aux2 As Integer                         'Auxiliar Vatriable No2
Dim Best As Integer                         'The Best Value
Dim c As Variant                            'c Joker Variable
```

```
Dim Clmn1 As Integer                    'The Number of Columns in Range1
Dim Clmn2 As Integer                    'The Number of Columns in Range2
Dim Data00(1080, 5) As Integer          'The Database Info
Dim Data01(1080, 5) As Integer          'Restructured Database Info
Dim Flag As Boolean                     'Flag 1
Dim Grph(60, 18) As Integer             'The Graph Info
Dim i As Integer                        'Counter i
Dim j As Integer                        'Counter j
Dim k As Integer                        'Counter k
Dim Mods As Integer                     'The Maximum Lectures per Hour
Dim M(60, 60) As Byte                   'Distance Matrix of Colors
Dim Min As Integer                      'The Minimum Rigidity
Dim NoClrs As Integer                   'Number of Allowed Colors in The Graph
Dim Optm As Integer                     'Optimum Rigidity
Dim p1 As Integer                       'Location Number 1
Dim p2 As Integer                       'Location Number 2
Dim r0 As Integer                       'Random Number 0
Dim r1 As Integer                       'Random Number 1
Dim r2 As Integer                       'Random Number 2
Dim Rnge1 As Range                      'Range of Graph's Information
Dim Rnge2 As Range                      'Range of the Matrix M
Dim Rw1 As Integer                      'The Number of Rows in Range1
Dim Rw2 As Integer                      'The Number of Rows in Range2
Dim Samp As Integer                     'Number of Candidates
Dim Str1 As String                      'The Graph's Information String
Dim Str2 As String                      'The Matrix M String
Dim Sum As Integer                      'The Sum Receipt
Dim Swap(60, 8) As Integer              'Swaps Info
Dim Tabu(1080, 1080) As Integer         'Tabu Information
Public Cntr As Integer                  'General Counter
Public Dev As Double                    'Target Deviation
Public Error As Double                  'Error Deviation
Public Iterations As Integer            'Number of Iterations
Public t1 As Integer                    'The Recency Period Allowed

Sub TabuSearch()

Load UserForm1
UserForm1.Show

End Sub

Sub GetData()

'This procedure gets the necessary information to run the program

Str1 = UserForm1.RefEdit1.Value
Erase Data00

Set Rnge1 = Range(Str1)
Clmn1 = Rnge1.Columns.Count
Rw1 = Rnge1.Rows.Count

i = 1
j = 1

For Each c In Rnge1
   Data00(i, j) = c.Value
   j = j + 1
   If j = Clmn1 + 1 Then
```

```
        j = 1
        i = i + 1
    Else
        'Do nothing
    End If
Next c

Str2 = UserForm1.RefEdit2.Value
Erase M

Set Rnge2 = Range(Str2)
Clmn2 = Rnge2.Columns.Count
Rw2 = Rnge2.Rows.Count

i = 1
j = 1

For Each c In Rnge2
    M(i, j) = c.Value
    j = j + 1
    If j = Clmn2 + 1 Then
        j = 1
        i = i + 1
    Else
        'Do nothing
    End If
Next c

End Sub

Sub FindOptimum()

'In this part we calculate the Minimum possible rigidity of the graph

NoClrs = Clmn2

i = 1
j = 1
Optm = 0

Do While i <= Rw1
    Data00(i, 5) = j
    j = j + 1
    If j < (NoClrs + 1) Then
        i = i + 1
    Else
        i = i + 1
        j = 1
    End If
Loop

c = 0

For i = 1 To NoClrs
    For j = 1 To Rw1
        If Data00(j, 5) = i Then
            c = c + 1
        Else
            'Do nothing
        End If
```

```
    Next j
    If c <= 1 Then
        'Do Nothing
    Else
        Optm = Optm + comb(c, 2)
        c = 0
    End If
Next i

End Sub

Sub StartGraph()

'The following instructions initialize the necessary values to start the cycle

Rnge1.Cells(0, Clmn1 + 1).Value = ""
Rnge1.Cells(0, Clmn1 + 1).Borders(xlEdgeTop).LineStyle = 0
Rnge1.Cells(0, Clmn1 + 1).Borders(xlEdgeBottom).LineStyle = 0

For i = 1 To Rw1
    Rnge1.Cells(i, Clmn1 + 1).Value = ""
Next i

Rnge1.Cells(1, Clmn1 + 3).Value = ""
Rnge1.Cells(2, Clmn1 + 3).Value = ""
Rnge1.Cells(3, Clmn1 + 3).Value = ""
Rnge1.Cells(4, Clmn1 + 3).Value = ""
Rnge1.Cells(1, Clmn1 + 4).Value = ""
Rnge1.Cells(2, Clmn1 + 4).Value = ""
Rnge1.Cells(3, Clmn1 + 4).Value = ""
Rnge1.Cells(4, Clmn1 + 4).Value = ""

Mods = CInt(UserForm1.TextBox1.Value)
Samp = CInt(UserForm1.TextBox2.Value)
Iterations = CLng(UserForm1.TextBox3.Value)
t1 = CInt(UserForm1.TextBox4.Value)
Dev = CDbl(UserForm1.TextBox5.Value)
Dev = Dev / 100

Erase Data01
Erase Tabu
Erase Grph
Flag = False

c = 1
k = 0

For i = 1 To Mods

    Sum = 0

    For j = 1 To Rw1
        If Data00(j, 2) = i Then
            Sum = Sum + 1
        Else
            'Do Nothing
        End If
    Next j

    k = k + (NoClrs - Sum)
```

```
    For j = 1 To NoClrs
        If j <= (NoClrs - Sum) Then
            Data01(c, 1) = c
            Data01(c, 2) = i
            Data01(c, 5) = j
            c = c + 1


        Else
            Data01(c, 1) = c
            Data01(c, 2) = Data00(c - k, 2)
            Data01(c, 3) = Data00(c - k, 3)
            Data01(c, 4) = Data00(c - k, 4)
            Data01(c, 5) = j
            c = c + 1
        End If
    Next j

Next i

For i = 1 To NoClrs
    For j = 1 To Mods
        For k = 1 To Mods * NoClrs
            If Data01(k, 2) = j And Data01(k, 5) = i Then
                Grph(i, j) = Data01(k, 1)
            Else
                'Do Nothing
            End If
        Next k
    Next j
Next i

c = 0

For i = 1 To Mods * NoClrs
    For j = 1 To Mods * NoClrs
        If j > i Then
            If Data01(i, 4) > 0 And Data01(j, 4) > 0 Then

                If Data01(i, 3) = Data01(j, 3) Then
                    If M(Data01(i, 5), Data01(j, 5)) = 0 Then
                        c = c + 100
                    End If
                End If

                If Data01(i, 5) = Data01(j, 5) Then
                    c = c + 1
                End If

            End If
        End If
    Next j
Next i

Best = c
Error = ((Best - Optm) / Optm)

End Sub
```

```
Sub SwapLocations()

Erase Swap
Erase Aux0

For i = 1 To Samp

   r0 = CInt((Mods - 1) * Rnd()) + 1
   r1 = CInt((NoClrs - 1) * Rnd()) + 1
   r2 = CInt((NoClrs - 1) * Rnd()) + 1

   If r1 > r2 Then
      Aux1 = r1
      Aux2 = r2
      r1 = Aux2
      r2 = Aux1
   End If

   p1 = ((r0 - 1) * NoClrs) + r1
   p2 = ((r0 - 1) * NoClrs) + r2

   Aux1 = Data01(Grph(r1, r0), 5)
   Aux2 = Data01(Grph(r2, r0), 5)
   Data01(Grph(r1, r0), 5) = Aux2
   Data01(Grph(r2, r0), 5) = Aux1

   c = 0

   For j = 1 To Mods * NoClrs
      For k = 1 To Mods * NoClrs
         If k > j Then
            If Data01(j, 4) > 0 And Data01(k, 4) > 0 Then

               If Data01(j, 3) = Data01(k, 3) Then
                  If M(Data01(j, 5), Data01(k, 5)) = 0 Then
                     c = c + 100
                  End If
               End If

               If Data01(j, 5) = Data01(k, 5) Then
                  c = c + 1
               End If

            End If
         End If
      Next k
   Next j

   Aux1 = Data01(Grph(r2, r0), 5)
   Aux2 = Data01(Grph(r1, r0), 5)
   Data01(Grph(r1, r0), 5) = Aux1
   Data01(Grph(r2, r0), 5) = Aux2

   Swap(i, 1) = r0
   Swap(i, 2) = r1
   Swap(i, 3) = r2
   Swap(i, 4) = p1
   Swap(i, 5) = p2
   Swap(i, 6) = c
   Swap(i, 7) = c + CInt(Round((Tabu(p2, p1) / Cntr) * c, 0))
```

```
   Swap(i, 8) = Tabu(p1, p2)

Next i

End Sub

Sub SelectSwap()

If Flag = False Then
   For i = 1 To Samp
      For j = 1 To Samp
         If Swap(i, 6) < Swap(j, 6) Then
            For k = 1 To 8
               Aux0(i, k) = Swap(j, k)
               Aux0(j, k) = Swap(i, k)
               Swap(i, k) = Aux0(i, k)
               Swap(j, k) = Aux0(j, k)
            Next k
         End If
      Next j
   Next i
End If

If Flag = False Then
   i = 1
   Do While Flag = False And i <= Samp
      If Swap(i, 6) < Best Then
         Min = i
         Flag = True
         Best = Swap(i, 6)
      Else
         If Cntr > Swap(i, 8) Then
            Min = i
            Flag = True
         End If
      End If
   i = i + 1
   Loop

End If

If Flag = False Then
   For i = 1 To Samp
      For j = 1 To Samp
         If Swap(i, 7) <= Swap(j, 7) Then
            For k = 1 To 8
               Aux0(i, k) = Swap(j, k)
               Aux0(j, k) = Swap(i, k)
               Swap(i, k) = Aux0(i, k)
               Swap(j, k) = Aux0(j, k)
            Next k
         End If
      Next j
   Next i
End If

If Flag = False Then
   i = 1
   Do While Flag = False And i <= Samp
      If Cntr > Swap(i, 8) Then
```

```
            Min = i
            Flag = True
        End If
        i = i + 1
    Loop

End If

If Flag = False Then
    Min = 1
    Flag = True
End If

End Sub

Sub UpdateValues()

Aux1 = Grph(Swap(Min, 2), Swap(Min, 1))
Aux2 = Grph(Swap(Min, 3), Swap(Min, 1))
Grph(Swap(Min, 2), Swap(Min, 1)) = Aux2
Grph(Swap(Min, 3), Swap(Min, 1)) = Aux1

Aux1 = Data01(Grph(Swap(Min, 2), Swap(Min, 1)), 5)
Aux2 = Data01(Grph(Swap(Min, 3), Swap(Min, 1)), 5)
Data01(Grph(Swap(Min, 2), Swap(Min, 1)), 5) = Aux2
Data01(Grph(Swap(Min, 3), Swap(Min, 1)), 5) = Aux1

If Swap(i, 2) <> Swap(i, 3) Then
    Tabu(Swap(Min, 4), Swap(Min, 5)) = Cntr + t1
    Tabu(Swap(Min, 5), Swap(Min, 4)) = _
    Tabu(Swap(Min, 5), Swap(Min, 4)) + 1
End If

Error = (Best - Optm) / Optm
Flag = False

End Sub

Sub Output()

Rnge1.Cells(0, Clmn1 + 1).Value = "Color"
Rnge1.Cells(0, Clmn1 + 1).HorizontalAlignment = xlCenter
Rnge1.Cells(0, Clmn1 + 1).Borders(xlEdgeTop).LineStyle = 1
Rnge1.Cells(0, Clmn1 + 1).Borders(xlEdgeBottom).LineStyle = 1

For i = 1 To Rw1
    For j = 1 To Mods * NoClrs
        If Data00(i, 3) = Data01(j, 3) Then
            If Data00(i, 4) = Data01(j, 4) Then
                Rnge1.Cells(i, Clmn1 + 1).Value = Data01(j, 5)
            End If
        End If
    Next j
Next i

Rnge1.Cells(1, Clmn1 + 3).Value = "Counter"
Rnge1.Cells(2, Clmn1 + 3).Value = "Best"
Rnge1.Cells(3, Clmn1 + 3).Value = "Rigidity"
Rnge1.Cells(4, Clmn1 + 3).Value = "Error"
Rnge1.Cells(1, Clmn1 + 4).Value = Cntr
```

```
Rnge1.Cells(2, Clmn1 + 4).Value = Best
Rnge1.Cells(3, Clmn1 + 4).Value = Swap(Min, 7)
Rnge1.Cells(4, Clmn1 + 4).Value = Error

End Sub

Private Function comb(n As Variant, r As Integer) As Long

Dim kfac As Long
Dim nfac As Long
Dim nkfac As Long

kfac = 1
nfac = 1
nkfac = 1

For k = 1 To n
   If k <= n Then
      nfac = nfac * k
   End If
   If k <= r Then
      kfac = kfac * k
   End If
   If k <= (n - r) Then
      nkfac = nkfac * k
   End If
Next k

comb = nfac / (kfac * nkfac)

End Function
```
