Revista de Matemática: Teoría y Aplicaciones 2013 **20**(2) : 215–230
cimpa – ucr issn: 1409-2433

# ALGORITMO DE BÚSQUEDA TABÚ PARA UNA VARIANTE DEL PROBLEMA DE COLORACIÓN

# TABU SEARCH ALGORITHM FOR A VARIATION OF THE COLORING PROBLEM

Mario Aboytes–Ojeda* Ana Lilia Laureano–Cruces†
Javier Ramírez–Rodríguez‡

*Received: 12/Mar/2012; Revised: 15/May/2013;*
*Accepted: 30/May/2013*

---

*Valle de Guadiana 110 Colonia Valle del Campestre, 37150 León Gto., México. (Posgrado en Ingeniería de Sistemas, Facultad de Ingeniería, Universidad Nacional Autónoma de México), México D.F., México. E-Mail: aboalfa80@yahoo.com

†Departamento de Sistemas, Universidad Autónoma Metropolitana, Unidad Azcapotzalco, Avenida San Pablo 180 Colonia Reynosa, 02200 México D.F., México; y Laboratoire d'Informatique d'Avignon, Université d'Avignon et des Pays de Vaucluse, France. E-Mail: clc@azc.uam.mx

‡Universidad Autónoma Metroplitana y Laboratoire d'Informatique d'Avignon, Université d'Avignon et des Pays de Vaucluse. Misma dirección que/*Same address as:* A.L. Laureano. E-Mail: jararo@azc.uam.mx

**Resumen**

El problema de coloración robusta generalizado (PCRG) resuelve problemas de horarios que consideran restricciones especiales. Al ser una generalización del problema de coloración robusta, que es a su vez una generalización del problema de coloración, el PCRG es entonces un problema NP-Completo, por lo que es necesario utilizar métodos aproximados para encontrar buenas soluciones en un tiempo de cómputo razonable. En este trabajo se presenta un algoritmo de búsqueda tabú para programar casos de 30 a 180 horas por semana, para algunos de ellos encuentra la solución óptima, en otros casos, la solución obtenida supera a la mejor solución conocida. También se presentan ejemplos de mayor tamaño a los conocidos, obteniendo resultados muy competitivos, lo que se puede verificar por la ausencia de conflicto entre clases.

**Palabras clave:** coloración robusta, problemas de horarios, heurísticas.

**Abstract**

The generalized robust coloring problem (GRCP) resolves timetabling problems that consider special constraints. As a generalization of the robust coloring problem, which is in turn a generalization of the coloring problem, the GRCP is then a NP-Complete problem, so it is necessary to use approximate methods to find good solutions in a reasonable computation time. This paper presents a tabu search algorithm to schedule cases from 30 to 180 hours per week, for some of them it found the optimal solution, in other cases, the solution obtained exceeds the best known solution. It also presents examples of greater size to the known, obtaining very competitive results, which can be verified by the absence of class conflict.

**Keywords:** robust coloring, timetabling problems, heuristics.

**Mathematics Subject Classification:** 05C15, 90C59, 68T20.

# 1 Introducción

La asignación de horarios es un problema muy común en las universidades, cada una tiene uno propio [3], [2], [1]. En Ramírez (2001) [9] se introdujo el problema de coloración robusta generalizado (PCRG), el cual resuelve problemas de horarios que consideran restricciones del tipo: dos eventos no pueden realizarse a la misma hora y debe haber al menos $d$ días entre dos clases de la misma materia. Se demostró que el problema es NP-Difícil, por lo que es necesario utilizar métodos aproximados para encontrar buenas

soluciones. En Lara et al. (2011) [7] se presenta un procedimiento glotón aleatorizado, GRASP, para programar cursos que requieren desde 30 hasta 120 horas por semana, en Pérez de la Cruz y Ramírez Rodríguez (2011) [8] se presenta un híbrido de un algoritmo genético con uno de búsqueda local que encuentra mejores soluciones para algunos de los mismos casos.

Dada una gráfica $G = (V, A)$, donde $V$ es un conjunto de vértices y $A$ un conjunto de aristas, una coloración de $G$ con $K$ colores es válida si los vértices unidos por una arista tienen color diferente. Si una gráfica se puede colorear con $k$ colores, se dice que es $k$-coloreable. Al menor valor de $k$ tal que $G$ es $k$-coloreable se le llama número cromático.

De acuerdo con Ramírez (2001) [9] y Yáñez y Ramírez. (2003) [11] se define la rigidez de la coloración $C$ como la suma de las penalizaciones de las aristas complementarias cuyos extremos están igualmente coloreados. Dada una gráfica $G = (V, A)$ con $|V| = n$, un entero positivo $k$ y una matriz de penalizaciones $P = \{p_{ij}, (i, j) \in \bar{A}\}$, el Problema de Coloración Robusta (PCR) encuentra una coloración válida $C : V \to \{1, 2, ..., k\}$, donde $k$ no es necesariamente el número cromático, que cuente con el menor grado de rigidez $R(C)$.

Dado un conjunto no vacío $B$, una distancia es una aplicación $d : B \times B \to \mathbb{R}$ verificando las siguientes propiedades:

- Simetría: $\forall x, y \in C, d(x, y) = d(y, x)$.
- No negatividad: $\forall x, y \in C, d(x, y) \geq 0$.
- Identidad: $\forall x \in C, d(x, x) = 0$.
- Desigualdad del triángulo: $\forall x, y, z \in C, d(x, z) \leq d(x, y) + d(y, z)$.

Dada una $k$-coloración $C$, $d$ es una distancia en el conjunto de colores $\{1, 2, ..., k\}$.

El PCR se define como sigue:

$$Min \quad R(C) = \sum_{\{i,j\} \in \bar{A}, C(i)=C(j)} p_{ij}$$
$$s.a \quad C(i) \neq C(j) \quad \forall \{i, j\} \in A.$$

El PCRG encuentra una coloración válida con un número fijo de colores que cuente con el menor número de violaciones a restricciones del tipo "debe haber al menos $d$ días entre dos eventos de un mismo tipo". También

se define en Ramírez (2001) [9] y Lara et al. (2011) [7] el $\bar{d}$-grado de rigidez generalizado de la $k$-coloración $(C)$, siendo $d \geq 0$, como la suma de las penalizaciones de las aristas complementarias cuyos extremos están pintados con colores que tienen una distancia menor o igual a $\bar{d}$:

$$R^{\bar{d}}(C) = \sum_{\{i,j\}\in\bar{A}, d(C(i)=C(j))\leq\bar{d}} p_{ij}$$

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

El PCRG trata de encontrar encontrar la coloración con menor grado de rigidez generalizada.

Se presenta un algoritmo de búsqueda tabú para el PCRG que, como Pérez de la Cruz y Ramírez Rodríguez (2011) [8], encuentra mejores soluciones para algunos casos reportados en Lara et al. (2011) [7] y como el algoritmo genético y caso reportado en Ramírez (2001) [9] encuentra la solución óptima.

El artículo está organizado como sigue: en la sección 2 se describe el algoritmo de búsqueda tabú, en la Sección 3 se presentan resultados computacionales y finalmente se presentan algunas conclusiones.

## 2 La búsqueda tabú para el PCRG

La búsqueda tabú (BT) es un algoritmo meta-heurístico que se distingue por el uso de una memoria adaptativa y de estrategias especializadas de procesamiento de información, fue desarrollado por F. Glover (1989) [4] y varios investigadores la han utilizado para encontrar buenas soluciones a ciertos problemas. 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 [6], [5].

La BT integra metodologías diseñadas para evitar óptimos locales y explorar nuevas regiones en el entorno de soluciones. El éxito relevante del método para problemas de optimización duros ha causado un brote considerable de nuevas aplicaciones en los últimos años. Entre sus aplicaciones se encuentran por ejemplo los problemas del agente viajero, asignación cuadrática, secuenciación de la producción y una variedad de problemas de diseño entre otros [10].

Los procedimientos del algoritmo trabajan bajo la suposición de que se puede crear un vecindario de soluciones que pueden ser alcanzadas desde

una posición inicial utilizando movimientos de permutación. Al realizar una permutación se intercambia una solución actual por otra del entorno de búsqueda de manera similar al algorimto de búsqueda local (BL).

En el algoritmo de BL se tienen tres pasos elementales:

1. Inicialización
   (a) Seleccionar una solución de arranque $xActual \in X$.
   (b) Almacenar la mejor solución actual conocida haciendo $xMejor = xActual$ y definiendo $MejorCosto = c(xMejor)$.
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.
3. Actualización
   Rehacer $xActual = xSiguiente$, y si $c(xActual) < MejorCosto$, ejecutar el paso 1(b). Luego volver al paso 2.

La búsqueda tabú puede interpretarse como una búsqueda en el entorno con mecanismos más elaborados ya que utiliza la información histórica para seleccionar el siguiente movimiento en lugar de hacerlo de manera aleatoria.

En el siguiente ejemplo tomado de Ramírez (2001) [9] y Lara et al. (2011) [7] se desea programar los cursos de un diplomado en donde las clases constan 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 a continuación:

| **Módulo** | **Materia** | **Clases por semana** |
|---|---|---|
| I | A,B,C | 3 |
| | K | 1 |
| II | D,E | 3 |
| | F,G | 2 |
| III | H,I,J | 2 |
| | L,M,N,O | 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 programadas 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 curso en el mismo intervalo de tiempo.

Para este ejemplo se utiliza el modelo de gráfica que fue introducido en [9]. Se asocia un vértice de la gráfica a cada clase $X_i$ de cada materia del programa $X \in \{A, B, \ldots, O\}$, con $1 \leq i \leq n_x$, 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 3 módulos.

## 2.1 Representación de la solución

Cada materia del curso tiene un determinado número de clases que se deben de impartir y que son representadas mediante los vértices de la gráfica, a cada una de ellas se les puede etiquetar de la siguiente manera:

| **Materia** | **Clase** | **Etiqueta** |
|---|---|---|
| A | 1 | 1 |
| A | 2 | 2 |
| A | 3 | 3 |
| B | 1 | 4 |
| B | 2 | 5 |
| B | 3 | 6 |
| ⋮ | ⋮ | ⋮ |
| O | 1 | 30 |

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:

| Día | Hora | Etiqueta |
|---|---|---|
| lunes | 1 | 1 |
| lunes | 2 | 2 |
| lunes | 3 | 3 |
| martes | 1 | 4 |
| martes | 2 | 5 |
| martes | 3 | 6 |
| ⋮ | ⋮ | ⋮ |
| viernes | 3 | 15 |

Entonces cada solución puede ser representada asignándole a cada uno de los vértices (clases) un color que representa la hora a la que será programada, por ejemplo:

| Clase (vértice) | Hora (color) |
|---|---|
| 1 | 1 |
| 2 | 7 |
| 3 | 13 |
| 4 | 2 |
| 5 | 8 |
| 6 | 14 |
| ⋮ | ⋮ |
| 30 | 1 |

## 2.2 Entorno de soluciones

Una representación adecuada para la búsqueda local define para cada solución $x \in X$, un conjunto asociado de vecinos, $N(x) \subset X$, llamado entorno o vecindario de $x$. Al igual que en la búsqueda local, la búsqueda tabú trabaja también con entornos de soluciones, y por tanto, se debe definir también su vecindario.

En esta sección se introduce la matriz $L_{r\times s}$ para definir el entorno de soluciones, con $n$ 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, el cual puede estar vacío o puede contener a una sola clase $X_i$ del programa. La matriz de locaciones es entonces:

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

Las locaciones existentes en cada fila de la matriz no pueden exceder el número de cursos $s$ dada la restricción de que no puede haber dos clases del mismo curso programadas a la misma hora. Para el problema ilustrativo que se presento anteriormente se tiene un vecindario como el que se muestra a continuación:

| **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 | $l_{15,1}$ | $l_{15,2}$ | $l_{15,3}$ |

La tabla es la estructura que almacena las clases del programa, cada celda de la tabla corresponde a una locación de la matriz $L_{r\times s}$, la cual puede estar vacía (cero para facilitar los cálculos) o puede contener a una sola clase del problema. La tabla anterior tiene 45 celdas de las cuales 30 serán ocupadas por las clases del programa y 15 quedarán disponibles. Una vez que la tabla tiene definidas sus dimensiones, el siguiente paso es el acomodo de las clases existentes en sus celdas, lo que corresponde a la generación de una solución inicial.

Para generar una solución inicial puede utilizarse algún algoritmo sencillo o hacerlo de manera aleatoria. En la figura que se muestra a continuación se puede observar una solución inicial generada mediante el acomodo de las clases en las últimas celdas de cada columna. En total la figura muestra una tabla con 45 celdas conformadas de la siguiente manera:

- 5 celdas vacías y 10 ocupadas en la primera columna.
- 5 celdas vacías y 10 ocupadas en la segunda columna.
- 5 celdas vacías y 10 ocupadas en la última columna.

| 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 con $l_{x,y} = Y_j$ y $l_{w,z} = X_i$. El movimiento puede modificar la estructura de la tabla mediante el intercambio del contenido de un par de celdas ubicadas en una misma columna. Como ejemplo de un movimiento sobre la tabla anterior se tiene: $l_{7,1} = A_2$ y $l_{4,1} = 0$ para obtener $l_{7,1} = 0$ y $l_{4,1} = A_2$.

En la búsqueda tabú es primordial el proceso de la memoria adaptativa para llevar el registro de los movimientos realizados y dirigir una búsqueda inteligente de una solución que satisfaga adecuadamente los requerimientos del problema.

## 2.3 Los movimientos recientes y su frecuencia

La historia $H$ de la búsqueda tabú utiliza una memoria que se basa en el registro de los movimientos que han ocurrido durante la búsqueda. El registro 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. Para simplificar la estructura de memoria y registrar los movimientos de permutación 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 $u_x = Y_j$ y $u_y = X_i$. La estructura de memoria para el registro de los movimientos de intercambio realizados en el ejemplo presentado se muestra en seguida:

| | $u_1$ | $u_2$ | $u_3$ | $u_4$ | $u_5$ | $u_6$ | $\cdots$ | $u_n$ |
|---|---|---|---|---|---|---|---|---|
| $u_1$ | | | | | | | | |
| $u_2$ | | | | | | | | |
| $u_3$ | | | | | | | | |
| $u_4$ | | | | | | | | |
| $u_5$ | | | | | | | | |
| $u_6$ | | | | | | | | |
| $\vdots$ | | | | | | | | |
| $u_n$ | | | | | | | | |

Los movimientos de permutación almacenados son a menudo usados en búsqueda tabú para imponer restricciones que evitan que sean elegidos movimientos que invertirían los cambios hechos por el algoritmo. Cuando se ejecuta un movimiento de $xActual$ a $xSiguiente$ ocurre un evento $e$ y se mantiene un registro para el evento de su movimiento inverso, que se denota por $\bar{e}$ y se utiliza para prevenir que se deteriore la búsqueda.

La parte de la tabla que se ubica en la esquina superior derecha, por encima de la diagonal, se utiliza para contener la información $FinTabu(e) = ite + t$, donde $ite$ es la iteración en la cual se realizó el intercambio y $t$ que representa el horizonte durante el cual se clasificará el movimiento como tabú.

El número de veces que cierto movimiento se ha realizado durante la búsqueda de la solución también se utiliza para imponer restricciones que evitan seleccionar un movimiento que ha sido muy utilizado por la BT. La parte inferior izquierda que se encuentra por debajo de la diagonal de la tabla, almacena la frecuencia con que dichos movimientos han ocurrido. Para registrar las medidas de transición en la memoria de frecuencia se utiliza $f_x = \#S(u_x = X_i \text{ a } u_y = X_i)$, incrementando su registro cada que se presenta dicho movimiento de intercambio .

Los intercambios de los contenidos de las celdas ocurren en la misma columna para prevenir colocar dos clases del mismo curso a una misma hora. Estos intercambios dan lugar a combinaciones que mantienen la restricción de programar clases del mismo curso en diferentes horas.

## 2.4 La calidad de la solución

Para medir la calidad de las soluciones obtenidas por la BT se utiliza la definición del $\bar{d}$-grado de rigidez de la gráfica:

$$R^{\bar{d}}(C) = \sum_{\{i,j\}\in\bar{A}, d(C(i)=C(j))\leq\bar{d}} p_{ij}$$

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

La calidad de la solución define el criterio de aspiración del algoritmo BT para el PCRG, cuando un movimiento que se clasifica como tabú genera una solución que supera en calidad a la mejor solución encontrada (menor grado de rigidez de la gráfica), entonces dicha clasificación se elimina y se realiza el movimiento de mejora. Cuando la calidad de la solución es aceptable entonces el algoritmo detiene su ejecución y arroja los resultados obtenidos.

## 2.5 Algoritmo de búsqueda tabú

Con el entorno de soluciones y los parámetros de lo reciente, frecuencia y calidad definidos se introduce a continuación el algoritmo a detalle:

1. Generar una solución inicial.
2. Calcular la rigidez inicial de la solución.
3. Crear una lista de candidatos con movimientos de permutación aleatorios.
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 $\phi$ iteraciones.
7. Generar los resultados.

Una vez que se genera una solución inicial se procede a evaluarla con la función de rigidez, obteniendo una rigidez inicial $R_0^{\bar{d}}(C)$. La evaluación se convierte en el valor $MejorRigidez$ en la iteración inicial $ite = 0$. Este valor será reemplazado por $R^{\bar{d}}(C)$ cuándo $R^{\bar{d}}(C) < MejorRigidez$. 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 $\mu$ candidatos que nos de variantes en la dirección de la búsqueda de la solución sin tener que explorar el entorno completo de soluciones. Los pasos para la generación de la lista de candidatos se muestra en seguida:

1. Escoger un curso $s_x$ aleatoriamente.
2. Seleccionar dos colores $r_{y1}$ y $r_{y2}$ al azar.
3. Obtener las posiciones de los colores seleccionados $u_i$ y $u_j$.
4. Realizar el intercambio de clases entre las posiciones.
5. Calcular la rigidez $R^{\bar{d}}(C)$ 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 $\mu$ veces.

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

$$R^{\bar{d}}(C)^* = R^{\bar{d}}(C) + \rho(\%f, R^{\bar{d}}(C)),$$

donde:

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

$$Mod(num, ite) = num - ite(\left\lfloor \frac{num}{ite} \right\rfloor)$$

$$num = f \times R^d(C).$$

La rigidez y la rigidez penalizada son parámetros que permiten la selección de un candidato de la lista generada por el algoritmo. Los pasos para seleccionar el candidato son:

1. Ordenar la lista de candidatos de menor a mayor grado de rigidez.

2. Seleccionar el cambio propuesto SI la rigidez $R^{\bar{d}}(C) < MejorRigidez$ o SI el número de iteración $\phi < FinTabu(e)$.

3. Repetir el paso número 2 HASTA cumplir alguna de las condiciones anteriores o HASTA haber recorrido los $\mu$ 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 $\phi < FinTabu(e)$.

7. Repetir el paso número 6 HASTA cumplir con la condición anterior o HASTA haber recorrido los $\mu$ candidatos.

8. Seleccionar el movimiento con menor rigidez penalizada $R^{\bar{d}}(C)^*$ 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.

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

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

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 $\alpha$ 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 de acuerdo al trabajo de Ramírez (2001) [9]. Se tiene entonces que:

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

en donde:

$$\begin{aligned} cph &= \alpha - Mod(\alpha, r)/r \\ Mod(\alpha, r) &= \alpha - r(\lfloor \alpha/r \rfloor). \end{aligned}$$

El $\%\Delta$ obtenido en cada iteración se comparará contra el error admisible $\%E$ y 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 todos los ejercicios que se presentan en la siguiente sección, logrando así la convergencia en soluciones de buena calidad.

## 3 Resultados

Los cursos que se programan en instituciones educativas varían por lo regular en su estructura y tamaño. En este trabajo se utilizan ejemplos que fueron empleados por Lara et al. (2011) [7] y por Pérez de la Cruz y Ramírez Rodríguez (2011) [8] para probar la eficacia del algoritmo BT. También se utilizó la notación que se define en Lara et al. (2011) [7] para generar dos nuevas instancias de mayor tamaño que no fueron consideradas en los trabajos mencionados. Se hizo un análisis de varianza y aunque no se encontraron diferencias significativas entre las soluciones de los métodos expuestos, las encontradas con BT son competitivas y en algunos casos mejores. Se presentan ejemplos de mayor tamaño a los conocidos con buenos resultados verificables por la ausencia de conflicto entre clases.

Se resolvieron 9 ejercicios en total para probar el buen funcionamiento del algoritmo incluyendo el ejemplo que se presentó durante el desarrollo del mismo. El resultado del problema ED4 nombrado así de acuerdo a la notación anteriormente mencionada es:

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

Se puede observar que en ninguna clase perteneciente a un mismo curso se repite la hora (color). También ninguna de las clases que pertenecen a una misma materia es programada el mismo día ni en días consecutivos.

Los resultados computacionales para diferentes instancias fueron:

| Clases | Ejemplo | Horas | $GRASP^*$ | $AG^{**}$ | $BT^{***}$ |
|---|---|---|---|---|---|
| 30 | ED4 | 15 | 1 | 1 | 1 |
| 30 | A42 | 15 | 0 | 0 | 0 |
| 60 | ECA864 | 20 | 0 | 0 | 0 |
| 60 | 976532 | 20 | 0 | 0 | 0 |
| 90 | EDDC96441 | 30 | 1 | 0 | 0 |
| 90 | DCB875322 | 30 | 0 | 0 | 0 |
| 120 | EEDCCBA87644 | 30 | 0 | 0 | 0 |
| 150 | EEDCCBA88776644 | 40 | - | - | 0 |
| 180 | EEDDCCBBAA88776644 | 40 | - | - | 0 |

La columna $GRASP^*$ contiene el número de clases que no pudieron ser acomodadas para que todos los intervalos de tiempo tuvieran el mismo número de clases utilizando el glotón aleatorizado, la columna $AG^{**}$ presenta la misma información utilzando el algoritmo genético híbrido y la última columna, $BT^{***}$, de igual manera con la búsqueda tabú. La BT muestra que logra la solución óptima en todos los casos, en el primer caso la solución es óptima de acuerdo a Pérez de la Cruz y Ramírez Rodríguez (2011) [8].

## 4 Conclusiones

La búsqueda tabú para PCRG que se presenta en este trabajo resulta ser eficiente en la exploración de buenas soluciones al problema de planificación de horarios con restricciones de dispersión en el tiempo. El algoritmo BT puede generar horarios de clases respetando las restricciones y aprovechando los recursos existentes en un tiempo razonable.

Existen áreas de oportunidad en el diseño del procedimiento aquí descrito que podrían mejorar significativamente su desempeño en la programación de horarios con restricciones especiales. Como ejemplo, se podría utilizar un algoritmo heurístico para generar una buena solución inicial en lugar de generarla aleatoriamente, esto podría ahorrar tiempo en la búsqueda de la solución. También se podría incluir una medida de influencia utilizando la distancia entre los colores para probablemente aumentar la velocidad de convergencia del algoritmo. Estas posibles mejoras pueden representar futuras líneas de investigación.

## Referencias

[1] Alvarez Valdéz, R.; Crespo, E.; Tamarit, J.M. (1997) "A tabu search algorithm to schedule university examinations", Qüestioó **21**(1 & 2): 201–215.

[2] de Werra, D. (1995) "Some combinatorial models for course scheduling", *PATAT'95*: 296–308.

[3] de Werra, D. (1996) "An introduction to timetabling", *Lecture Notes in Computer Science* **1153**, Springer: 296–308.

[4] Glover, F. (1989) "Tabu search - Part I", *INFORMS Journal on Computing* **1**: 190–206.

[5] Glover, F.; Laguna, M. (1997) *Tabu search*. Kluwer Academic Publishers, Norwell MA, U.S.A.

[6] Glover, F.; Melián-Batista, B. (2003) "Búsqueda tabú", *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial* **7**(19): 29–48.

[7] Lara Velázquez, P.; López Bracho, R.; Ramírez Rodríguez, J.; Yáñez, J. (2011) "A model for timetabling problems with period spread constraints", *JORS* **62**(1): 217–222.

[8] Pérez de la Cruz, C.; Ramírez Rodríguez, J. (2011) "Un algoritmo genético para un problema de horarios con restricciones especiales", *Rev. Mat. Teor. Aplic.* **18**(2): 215–229.

[9] Ramírez Rodríguez, J. (2001) *Extensiones del Problema de Coloración de Grafos*. Tesis Doctoral, Universidad Complutense de Madrid, Madrid, España.

[10] Schaerf, A. (1996) "Tabu search techniques for large high-school timetabling problems", *IEEE Transactions on Systems, Man, and Cybernetics*: 363–368.

[11] Yañez, J.; Ramírez, J. (2003) "The robust coloring problem", *European Journal of Operational Research* **148**: 546–558.