ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ciencias Naturales y Matemáticas

TEMA

Diseño e implementación de una heurística basada en la metodología GRASP para el problema de coloración de grafos aplicado a la calendarización de exámenes en una institución educativa.

TESIS DE GRADO

Previo a la obtención del Título de:
Magister en Investigación Matemática

Presentado por
Erwin Delgado Bravo

Guayaquil-Ecuador
2013

Agradecimiento

A la Santísima Trinidad, por todas las bendiciones y alegrías que me ha prodigado. A mi familia que me ha dado ánimo para alcanzar mis objetivos. A todas las personas que de alguna u otra manera colaboraron en la realización de esta tesis, en especial al Ph.D Francisco Vera y al M.Sc. Xavier Cabezas.

Dedicatoria

Que dificil es plasmar en un papel, el cúmulo de sentimientos que ocurren en mi pensamiento, con tan poco espacio con gran valor emocional. Puedo afirmar con certeza que mientras escribo estas palabras hacia Ustedes, siento que ocupo un lugar privilegiado dentro del paraiso. Para mis hijos Toñito y Carlitos y mi esposa Malena.

Declaración Expresa

La responsabilidad del contenido de esta Tesis de Posgrado, me corresponde exclusivamente; el patrimonio intelectual de la misma a la FCNM (Facultad de Ciencias Naturales y Matemáticas) de la ESPOL (Escuela Superior Politécnica Del Litoral).

Erwin Joffre Delgado Bravo

TRIBUNAL DE GRADUACIÓN

Oswaldo Valle Sánchez. M. Sc.
PRESIDENTE DEL
TRIBUNAL

Francisco Vera Alcívar. Ph. D.
DIRECTOR DE TESIS

Luis Rodríguez Ojeda M. Sc.
VOCAL

Índice general

Índice de figuras

Índice de tablas

Resumen

Una de las tareas que enfrentan las instituciones educativas de educación superior cada año, es la planificación de los horarios de clases y exámenes. Su dificultad radica en que diversas restricciones operativas surgen en el momento de la planificación. Específicamente, en el caso de la calendarización de exámenes, generalmente se intenta elaborarlos considerando diversas restricciones como por ejemplo que los mismos deben ser receptados en uno y sólo un período de tiempo y aula previamente definida, la misma que debe admitir un número máximo de estudiantes. De igual manera, la calidad de los calendarios está basado en promover aquellos que impidan que estudiantes rindan más de un examen en un mismo día, entre otras restricciones operativas.

Dada la naturaleza descrita anteriormente, la calendarización de exámenes pertenece al conjunto de problemas de optimización combinatoria categorizado NP-Duro por lo que resulta complejo resolverlo por métodos exactos. La ventaja es que la calendarización de exámenes es un problema operativo por lo que bastaría con obtener soluciones factibles de gran calidad, no necesariamente la óptima, en tiempos computacionales razonables. Uno de los métodos utilizados para el efecto, es la construcción de heurísticas basadas en metaheurísticas por la fortaleza en la exploración inteligente en el espacio de soluciones.

Con base en lo anterior, en el presente trabajo se desarrollará un algoritmo heurístico basado en la metodología GRASP el mismo que se lo aplicará en la confección de horarios de exámenes sujeto a un conjunto de restricciones de diversa índole.

El presente documento se lo ha dividido en cinco capítulos. En el capítulo uno se desarrolla una breve explicación del problema de calendarización de exámenes, así como

se definen los objetivos generales y específicos del presente trabajo.

En el capítulo dos se presenta el marco teoríco que fundamenta el proceso investigativo del presente trabajo. Se establecen las formulaciones matemáticas tanto para el problema de coloración de grafos, así como el de calendarización de exámenes.

En el capítulo tres se define claramente cada etapa del proceso de diseño del algoritmo propuesto, desde la fase de construcción de la solución inicial, así como del proceso de búsqueda local. Se hace énfasis del proceso de calibración de parámetros, utilizando para ello diversas instancias[1]. De igual manera, se presentan los resultados al aplicar el algoritmo con los parámetros ya calibrados y las comparaciones pertinentes con otro algoritmo, así como los valores óptimos.

En el capítulo cuatro se aplica el algoritmo propuesto en la confección de horarios factibles de exámenes en un semestre específico de una carrera de una institución educativa superior.

Por último, en el capítulo cinco se presentan diversas conclusiones y recomendaciones para futuros trabajos de investigación referentes al tema.

[1] http://mat.gsia.cmu.edu/COLOR/instances.html#XXLEI

Capítulo 1

Marco teórico

1.1. Descripción del problema

Una de las tareas que enfrentan cada año las instituciones educativas de educación superior, es la planificación de los horarios de clases y exámenes. Su dificultad radica en que diversas restricciones operativas surgen en el momento de su construcción. Específicamente, en el caso de la calendarización de exámenes, generalmente se intenta elaborarlos considerando diversas restricciones como por ejemplo que los mismos deben ser administrados en uno y sólo un período de tiempo y aula previamente definida sin exceder la capacidad de la misma.

Considerando lo anterior, y el hecho de que la institución educativa objeto del presente estudio posee un gran número de estudiantes registrados hace que este problema sea complejo de resolver por métodos exactos. La ventaja, es que la calendarización de exámenes es un problema operativo por lo que bastaría con obtener soluciones factibles de gran calidad, no necesariamente la óptima, en tiempos computacionales razonables. Una de los métodos utilizados para el efecto, es las construcción de heurísticas basadas en metaheurísticas por la fortaleza en la exploración inteligente en el espacio de

soluciones.

1.2. El problema de Calendarización de Exámenes

Considere un conjunto de N exámenes $E = \{e_1, e_2, ..., e_N\}$, los mismos que deben ser planificados en algún período $p \in \{p_1, p_2, ..., p_m\}$

Un calendario factible es aquel que satisface las siguientes restricciones fuertes:

- Todos los exámenes planificados son asignados a algún período de tiempo
- Todos los exámenes planificados son asignados en alguna aula.
- Ningún estudiante podrá rendir más de un examen en un mismo período de tiempo.
- En cada período de tiempo, no puede asignarse un aula para administrar dos o más exámenes
- La capacidad de las aulas no debe ser excedida.
- Alguna otra restricción operativa definida por la organización.

La calidad de un calendario factible está definido por el valor de la función objetivo que, entre otras cosas, penaliza alguna de las siguientes restricciones suaves:

- Materias de un mismo semestre no pueden ser planificadas en el mismo día
- A cada estudiante se le administrará máximo un examen por día.

- En caso de que a un estudiante se le administre más de un examen por día, estos no deben ser en períodos consecutivos.

- Alguna otra restricción operativa definida por la organización.

1.2.1. Formulación matemática del Problema de Calendarización de Exámenes

A continuación se presenta un modelo de programación no lineal presentado por McCollum B. et al [1] que permite encontrar una solución óptima al problema de calendarización de exámenes con la mínima penalización por violación de las restricciones suaves.

Conjuntos y parámetros

I) Exámenes:

a) E: Conjunto de exámenes.

b) se_i: Número de estudiantes registrados en el examen $i \in E$

II) Estudiantes:

a) S: Conjunto de estudiantes

b) $t_{is} = \begin{cases} 1, & \text{si el estudiante } s \text{ debe rendir el examen } i \\ 0, & \text{si no} \end{cases}$

III) Aulas

a) R conjunto de aulas

b) s_r: Capacidad del aula $r \in R$

IV) períodos

a) P : Conjunto de períodos

b) $y_{pq} = \begin{cases} 1, & \text{si los periodos } p \text{ y } q \text{ se encuentran en el mismo día} \\ 0, & \text{si no} \end{cases}$

c) H^{aft} : Conjunto de pares de exámenes. $\forall \quad (e_1, e_2) \in H^{aft}$ examen e_1 debe receptarse estrictamente despúes del examen e_2.

d) H^{coin} : Conjunto de pares de exámenes. $\forall \quad (e_1, e_2) \in H^{coin}$ los exámenes e_1 y e_2 deben ser administrados en el mismo período.

e) H^{excl} : Conjunto de pares de exámenes. $\forall \quad (e_1, e_2) \in H^{excl}$ los exámenes e_1 y e_2 no deben ser administrados en el mismo período.

V) Penalizaciones

a) w^{2R}: penalización por exámenes administrados en períodos consecutivos.

b) w^{2D}: penalización por exámenes administrados en el mismo día.

Variables

I) $xp_{ip} = \begin{cases} 1, & \text{si el examen } i \text{ debe programarse en el período } p \\ 0, & \text{si no} \end{cases}$

II) $xr_{ir} = \begin{cases} 1, & \text{si el examen } i \text{ debe programarse en el aula } r \\ 0, & \text{si no} \end{cases}$

III) $C2R_s$ Variable que indica el número de ocasiones que el estudiante s tiene dos exámenes administrados en períodos consecutivos en un día.

IV) $C2D_s$ Variable que indica el número de ocasiones que el estudiante s tiene dos exámenes planificados en un día.

Formulación Matemática

$$\min \ z = \sum_{s \in S} w^{2R} C2R_s + w^{2D} C2D_s \tag{1.1}$$

sujeto a:

$$\sum_{r \in R} xr_{ir} = 1, \ \ \forall i \in E \tag{1.2}$$

$$\sum_{p \in P} xp_{ip} = 1, \ \ \forall i \in E \tag{1.3}$$

$$se_i xp_{ip} xr_{ir} \leq s_r, \ \ \forall p \in P, \ \ \forall i \in E, \ \ \forall r \in R, \tag{1.4}$$

$$\sum_{i \in E} t_{is} xp_{ip} \leq 1, \ \ \forall p \in P, \ \ \forall s \in S \tag{1.5}$$

$$xp_{ip} + xp_{jq} \leq 1, \ \ \forall p, q \in P p \leq q, \ \ \forall (i,j) \in H_{aft} \tag{1.6}$$

$$xp_{ip} = xp_{jq}, \ \ \forall p \in P, \ \ \forall (i,j) \in H_{coin} \tag{1.7}$$

$$xp_{ip} + xp_{jq} \leq 1, \ \ \forall p \in P, \ \ \forall (i,j) \in H_{excl} \tag{1.8}$$

$$C2R_s = \sum_{\substack{i,j \in E \\ i \neq j}} \sum_{\substack{p,q \in P, \\ q = p+1, \\ y_{pq} = 1}} t_{is} t_{js} xp_{ip} xp_{jq} \tag{1.9}$$

$$C2D_s = \sum_{\substack{i,j \in E \\ i \neq j}} \sum_{\substack{p,q \in P \\ q > p+1 \\ y_{pq} = 1}} t_{is} t_{js} xp_{ip} xp_{jq} \tag{1.10}$$

$$xp_{ip}, xr_{ir} \in \{0,1\} \tag{1.11}$$

$$C2R_s, C2D_s \in Z; C2R_s \geq 0; C2D_s \geq 0 \ \ \forall s \in S \tag{1.12}$$

La ecuación (1.1) establece la función objetivo la cual es una combinación lineal del número de estudiantes que tienen exámenes consecutivos, así como exámenes que se

toman en el mismo día. La restricción (1.2) establece que cada examen sea administrado en alguna aula. La restricción (1.3) establece que cada examen sea administrado en algún período. La restricción (1.4) especifica que ningún examen sea administrado en un aula con capacidad menor al número de alumnos que deben rendir la prueba. La restricción (1.5) especifica que cada estudiante en cualquier período rinda a lo mucho un examen. la restricción (1.6) especifica el cumplimiento de que para cada par ordenado de exámenes que pertenezcan a H^{aft}, uno debe estar planificado estrictamente despúes del otro. De manera similar se analiza las restricciones (1.7) y (1.8). Las ecuaciones (1.10) y (1.9) permiten determinar el número de estudiantes que tienen planificado más de un examen en el mismo día, así como los que rinden dos exámenes consecutivos. Por último las restricciones (1.11) y (1.12) se refieren al caracter de las variables declaradas en el modelo.

Una de las aproximaciones utilizadas para obtener soluciones factibles de buena calidad pero no óptimas es por medio de la utilización de métodos que permite resolver el problema de coloración de grafos. Éste es el enfoque que se ha considerado en el presente proyecto.

1.3. El problema de coloración de grafos

Dado un grafo $G = (V, E)$ donde V es un conjunto de vértices y $E = \{(v_i, v_j) : v_i, v_j \in V, v_i \neq v_j\}$ un conjunto de arcos definidos en G, una coloración propia de G es una función $f : V \rightarrow [1, k]$ tal que $\forall (a, b) \in E$ se cumple que $f(a) \neq f(b)$, es decir, asigna a cada vértice de G un color diferente al asignado a cualquier otro vértice adyacente.

En este contexto, el Problema de Coloración de Grafos, PCG, consiste en determinar el mínimo número k de colores diferentes con el objeto de realizar una coloración propia de G. A este mínimo valor de k se le conoce como número cromático de G, denotado por $\chi(G)$.

Como se puede observar en la figura 1.1, para cada arco definido en el grafo dado, se observa que sus nodos adyacentes están coloreados con colores diferentes.

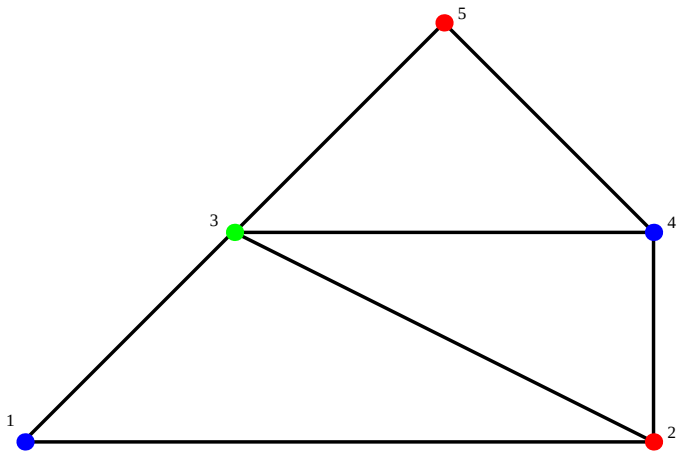

Figura 1.1: Ejemplo de una coloración propia de un Grafo G

1.3.1. Formulación matemática del Problema de Coloración de Grafos

A continuación, se presenta un modelo en programación lineal que permite encontrar la solución óptima al problema de coloración de grafos.

Índices

Sean $G = (V, E)$ un grafo no dirigido y C un conjunto de colores. Se establece lo siguiente:

i, k: son índices referidos al conjunto de vértices, mientras que

j: es un índice referido al conjunto de colores

Variables

$$x_{ij} = \begin{cases} 1, & \text{si el vértice } i \text{ es coloreado con el color } j \\ 0, & \text{si no} \end{cases}$$

Formulación Matemática

$$\min \ z = \sum_i \sum_j j x_{ij} \tag{1.13}$$

sujeto a:

$$\sum_j x_{ij} = 1, \quad \forall i \in V \tag{1.14}$$

$$x_{ij} + x_{kj} \leq 1, \quad \forall (i,k) \in E \quad \forall j \in C \tag{1.15}$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i,j) \in E \tag{1.16}$$

La ecuación (1.13) representa la función objetivo, la cual permite minimizar el número de colores utilizados para realizar una coloración propia. La restricción (1.14) permite asegurar que cada vértice sea coloreado por algún color. La restricción (1.15) permite

garantizar que para cada arco definido en el grafo, sus vértices adyacentes sean coloreados con colores distintos. Por último, la restricción (1.16) indica la naturaleza de las variables de decisión.

Precisamente, una de las aplicaciones del problema de coloración de grafos es el problema de calendarización de exámenes. En este problema, se realiza una abstracción matemática del mismo por medio de la utilización de un grafo $G = (V, A)$ en la que el conjunto V está formado por todas las materias planificadas en un período académico y $A = \{(i, j) : i$ y j son dos materias que no pueden ser planificadas el mismo día$\}$. La relación entre el PCG y la calendarización de exámenes se hace evidente en el hecho de que si dos materias no pueden ser planificadas el mismo día (i.e cada arco del grafo G), se les debe asignar distintos días para su evaluación (i.e sus vértices adyacentes deberán ser coloreados con colores diferentes).

1.4. Revisión del estado del arte

Uno de los enfoques utilizados para resolver el PCG es por métodos exactos. Así, Méndez I. et al [7] propone un modelo de programación entera, el mismo que es resuelto por un algoritmo de planos cortantes. De igual manera, Bodlaender H. et al [9] desarrolla un modelo exacto incorporando memoria polinomial mientras que Lucet C. et al [6] propone un método exacto basado en una descomposición lineal del grafo. Sin embargo estos métodos tienen limitaciones por la cardinalidad del conjunto de vértices del grafo.

Precisamente, el PCG es uno de los problemas de optimización combinatoria, categorizado en el trabajo desarrollado por Du D. et al [2] como NP-duro, por tal motivo

múltiples heurísticas se han diseñado con el objeto de obtener soluciones factibles de buena calidad en tiempos de ejecución adecuados. Uno de ellos es el realizado por Brélaz, D. [12] quién implementó un algoritmo glotón, ordenando en primer lugar los vértices de acuerdo al grado de adyacencia y asignando en cada iteración un color cada vértice obteniendo así una solución factible. De igual manera, Cédric A. et al [4], desarrollaron una búsqueda local en un vecindario variable. Uno de los trabajos a destacar, dentro de la categoria de métodos de búsqueda local, es el propuesto por Johnson D. et al [10], en el cual se diseña una búsqueda local con una función de evaluación de la coloración de un grafo, la misma que está en términos del tamaño de cada partición del conjunto de vértices, así como del número de arcos que violen la condición de coloración.

Dentro de los trabajos que se fundamentan en algoritmos evolutivos, podemos destacar los realizados por Lu Z. et al [3], quien desarrolló un algoritmo memético para este problema, resolviendo iterativamente el k-coloring desde un valor de k, disminuyéndolo hasta que no exista solución factible. Este algoritmo es una combinación de un algoritmo genético con una búsqueda tabú, incorporando un operador de «cruce adaptativo»; y el realizado por Eiben A. et al [8] que desarrolla un algoritmo evolutivo para la determinación de buenas soluciones factibles. Incorpora un «mecanismo adaptativo» que permite cambiar, en cada iteración, de la función de evaluación de una solución del problema. Por otro lado, A. Hertz et al [5], diseña un algoritmo fundamentado en la búsqueda tabú, la misma que la comparó con los resultados obtenidos por algunos trabajos desarrollados con recocido simulado observando un mayor desempeño para instancias de gran cardinalidad.

1.5. Metaheurísticas

En su definición clásica, las metaheurísticas son métodos de solución que integran procedimientos de búsqueda local y estrategias de alto nivel que permiten escapar de óptimos locales y realizar una «búsqueda robusta» en el espacio de soluciones [14]. Precisamente, la exploración del espacio de soluciones está ligada a procedimientos que permiten definir estructuras de vecindarios que son obtenidos por la transición de una solución a otra [14]. De aquí que el procedimiento utilizado para obtener soluciones a un problema permiten establecer el grado en la que los vecindarios son explorados.

Entre las metaheurísticas más utilizadas se podrían citar, la búsqueda local, búsqueda tabú, algoritmos genéticos, búsqueda dispersa, recocido simulado, redes neuronales, entre otras.

Generalmente, las metaheurísticas son utilizadas en problemas de optimización combinatoria cuyas soluciones no siempre pueden obtenerse por métodos exactos o si se las obtiene generalmente es a un gran costo computacional. Por ello, una de sus fortalezas es que permiten obtener soluciones de gran calidad a un costo computacional razonable en comparación con los métodos exactos de solución. Sin embargo, una de las mayores desventaja de utilizar las metaheurísticas es que éstas no proveen un certificado de optimalidad de las soluciones encontradas.

En el diseño de las metaheurísticas, dos criterios contradictorios deberán ser considerados [15]. Por un lado la exploración del espacio de búsqueda (diversificación) y la explotación de las mejores soluciones encontradas (intensificación). En la intensifica-

ción, regiones prometedoras son exploradas con el objeto de encontrar mejores soluciones mientras que en la diversificación se prentende explorar regiones no visitadas [15].

En el presente trabajo, se implementará un heurística basada en la metaheurística GRASP, para el problema de coloración de grafos.

1.5.1. GRASP

El GRASP (Greedy Randomized Adaptive Search Procedure) es una metaheurística en la que cada iteración consiste de dos etapas: construcción y búsqueda local [13]. El objetivo de la fase de construcción es la determinación de una solución factible la cual es mejorada en la fase de búsqueda local, lo que se muestra en el seudocódigo mostrado a continuación, considerando un problema de minimización:

Algoritmo 1 GRASP

```
1: procedure GRASP(MaxIter)
2:     for k=1,...,MaxIter do
3:         Seed=Reloj
4:         Solución1 ← EtapaContrucción(Seed)
5:         Solución ← BúsquedaLocal(Solución1)
6:         if Solución<BestSolución then
7:             BestSolución=Solución
8:         end if
9:     end for
10:    return BestSolución
11: end procedure
```

El fundamento de la fase de construcción de una solución factible es un algoritmo glotón no determinístico, en la que iterativamente se incorpora un elemento a una solución parcial. El elemento a ser incorporado en la solución parcial es escogido aleatoriamente desde una lista RCL, la cual está formada por aquellos elementos que produzcan los más bajos costos incrementales en la función objetivo. Luego que el elemento seleccionado es incorporado en la solución parcial, esta es actualizada y se procede a iterar nuevamente. A continuación se muestra el seudocódigo de la fase de construcción de la solución inicial:

Algoritmo 2 Fase de construcción

```
1: procedure CONSTRUCCIÓN
2:     Solución ← φ
3:     Evaluar el costo incremental de todos los elementos candidatos
4:     while Solución no está completa do
5:         Seed=Reloj
6:         Construir la lista de candidatos (RCL)
7:         Seleccionar un elemento s aleatoriamente de la lista RCL.
8:         Solución=Solución ∪ {s}.
9:         Reevaluar el costo incremental
10:    end while
11:    return Solución
12: end procedure
```

La solución generada en el esquema anterior no necesariamente es óptima por lo que usualmente se aplica un algoritmo de búsqueda local, en la que en cada iteración la solución actual es reemplazada por otra que pertenece a su vecindario y que mejore la función objetivo hasta que no sea posible tal mejora. A continuación se muestra el seudocódigo de la fase de búsqueda local.

Algoritmo 3 Búsqueda Local

```
1: procedure BúsquedaLocal(solución)
2:     while solución no sea óptima localmente do
3:         Encontrar s' ∈ N(solución) con f(s') < f(solución)
4:         solución ← s'
5:     end while
6:     return solución
7: end procedure
```

En este contexto, en el capítulo siguiente se explicará en detalle cada una de las componentes del algoritmo GRASP aplicado al problema de coloración de colores.

1.6. Objetivo General

Realizar la calendarización de exámenes en una institución educativa de educación superior.

1.7. Objetivos específicos

- Diseñar un algoritmo basado en la metodología GRASP para la resolución del problema de coloración de grafos.
- Aplicar un algoritmo basado en la metología GRASP para el problema de calendarización de exámenes en una institución educativa de educación superior.

Capítulo 2

Diseño de heurística basada en la metodología GRASP para el Problema de Coloración de Grafos

Como se ha mencionado anteriormente, el problema de coloración de grafos es un problema NP-duro, por lo que su resolución en problemas de gran escala no es posible realizarlas de forma exacta. Por ello, se suelen utilizar algoritmos basados en las metaheurísticas que permiten obtener soluciones de gran calidad a un costo computacional razonable. En este contexto, se ha diseñado un algoritmo basado en la metodología GRASP para el problema de coloración de grafos.

2.1. Diseño de fase de construcción

El objetivo principal en la fase de construcción, es la determinación de una solución factible del problema de coloración de grafos. Esta solución se la ha de determinar por medio de un algoritmo glotón no determinístico. Para tal efecto, se realizó una adaptación al algoritmo desarrollado por Brélaz, D. [12] para el problema de coloración

de grafos. A continuación se presenta el seudocódigo de la fase de contrucción de una solución factible al problema de coloración de grafos.

Algoritmo 4 Fase de constucción

1: **procedure** CONSTRUCCIÓN
2: solución $\leftarrow \phi$
3: Ordenar los vértices de acuerdo a su grado en orden decreciente.
4: Construir la lista de candidatos restringidas RCL definida por $\{v \in V | grad(v) \geq C_{min} + \alpha(C_{max} - C_{min})\}$, donde C_{max} y C_{min} son el máximo y mínimo grado de entre todos los vértices.
5: Seleccionar y colorear aleatoriamente un vértice s de la lista RCL
6: **while** $|solución| \neq |V|$ **do**
7: Seed=Reloj
8: Ordenar los vértices no coloreados de acuerdo a su grado de saturación en orden decreciente.
9: Construir la lista de candidatos restringidas RCL definida por $\{v \in V | grad(v) \geq C_{min} + \alpha(C_{max} - C_{min})\}$, donde C_{max} y C_{min} son el máximo y mínimo grado de saturación de entre todos los vértices no coloreados.
10: Seleccionar y colorear aleatoriamente un vértice s de la lista RCL, utilizando el mínimo número de colores para obtener una solución factible parcial.
11: solución $\leftarrow$ solución $\cup \{s\}$
12: **end while**
13: **return** *solución*
14: **end procedure**

Antes de explicar en detalle la adaptación realizada, es necesario introducirnos en la siguiente definición establecida por Brélaz, D. [12]

Definicion **2.1.** *Sea G un grafo simple y C una coloración parcial de G. Se define al grado de saturación de un vértice como el número de diferentes colores asignados a sus vértices adyacentes.*

De igual manera es importante destacar la siguiente definición establecida en [16]:

Definicion **2.2.** *Sea $G = (V, E)$ un grafo no dirigido o multígrafo. Para cada vértice $v \in V$, el grado de v denotado por $deg(v)$ es el número de arcos que son incidentes con v.*

2.1.1. Criterio glotón

Una de las componentes del algoritmo GRASP es la construcción de un algoritmo glotón. Diversos enfoques pueden ser consideradas para el efecto. Por ejemplo, se podría priorizar el grado de los vértices en el momento de colorear un vértice. En el presente trabajo inicialmente se procede a ordenar de forma decreciente los vértices de acuerdo a su grado. Este ordenamiento nos permite contruir una lista de «buenos » candidatos a ser considerados en la siguiente iteración. Sin embargo, este criterio cambia a partir del momento en que se va a colorear el segundo vértice de la solución parcial ya que el ordenamiento se lo realiza con base en los grados de saturación de los vértices no coloreados. Esta idea surge del hecho de que vértices de un número elevado de arcos incidentes se los debe colorear primero con el objeto de minimizar el número de colores utilizados para realizar una coloración propia.

2.1.2. Longitud de la lista RCL

Una de las características del algoritmo GRASP es que la construcción de la solución factible no es determinística ya que en cada iteración se incorpora un vértice a la solución parcial de forma aleatoria, el mismo que pertenece a una lista, denominada RCL, de buenos candidatos. Diversas estrategias pueden ser consideradas para la determinación de la longitud de la lista de candidatos; así, puede considerarse una lista con longitud fija

en todas las iteraciones o de longitud variable. Por ejemplo, sean C_{max} y C_{min} el máximo y mínimo grado de saturación de entre todos los vértices (o el grado de los vértices) entonces se construye el conjunto simbolizado como RCL de la siguiente manera:

$$RCL = \{v \in V | grad(v) \geq C_{min} + \alpha(C_{max} - C_{min})\} \tag{2.1}$$

donde α es un parámetro a considerar.

2.1.3. Elección del próximo vértice a ser coloreado

Luego de construida la lista RCL formada por los vértices considerados buenos candidatos y que no han sido coloreados, se procede a escoger aleatoriamente un vértice v de esta lista. Debido a que el GRASP debe realizarse un número de iteraciones simbolizado por MAX_ITER, la función generadora de números aleatorios uniformes está en términos de una variable entera $SEED$ que cambiará en cada iteración para garantizar la exploración de otras regiones. Al vértice v escogido aleatoriamente se le asignará un color que minimice el número de colores utilizados para realizar una coloración parcial del grafo y que garantice la factibilidad del mismo.

Este procedimiento se lo realizará iterativamente hasta que todos los vértices sean coloreados, obteniendo así una solución factible inicial.

Sin embargo, no necesariamente la solución factible obtenida en la etapa de construcción es la óptima, por lo que se hace imprescindible un algoritmo de mejora de la solución actual, objetivo que se cumple con la fase de búsqueda local.

2.2. Diseño de fase de búsqueda local

Como se ha explicado anteriormente, la fase de búsqueda local tiene por objeto mejorar una solución actual obteniendo así, en cada iteración, una solución de mejor calidad.

En este contexto, la fase de búsqueda local empieza con una solución inicial dada por la fase de construcción.

2.2.1. Vecindario de una solución

Uno de los requerimientos de cualquier búsqueda local es la forma de construcción de vecindades a una solución actual. En este sentido, una solución vecina a la actual es obtenida al colorear cualquier vértice «escogido» aleatoriamente por un color diferente, sea éste uno ya utilizado o por medio de otro color.

2.2.2. Evaluación de una solución

Conforme a la construcción del vecindario de una solución cualquiera, es evidente que al realizar un intercambio de colores podría generar soluciones no factibles, es decir asignar a un vértice un color que viole la restricción que cualquier vértice adyacente sea coloreado de la misma manera. En tal caso, se debería establecer una penalidad con el objetivo de promover coloraciones que impidan tal evento.

Sean $C = \{C_1, C_2, ..., C_k\}$ una partición de colores en el grafo $G = (V, E)$ y E_i el conjunto formado por los arcos que tienen sus vértices adyacentes en la partición $C_i,\ \forall i = 1, 2, ..., k$ entonces el valor de la coloración C, definida por Johnson D. et al [10], está dada por :

$$f(C) = -\sum_{i=1}^{k} |C_i|^2 + \sum_{i=1}^{k} |C_i||E_i| \tag{2.2}$$

Un hecho a considerar en la ecuación 2.2 es que esta función de evaluación promueve la creación de particiones de cardinalidad elevada, penalizando aquellas que contenga un elevado número de arcos que violen las restricciones suaves dadas. Adicionalmente se debe destacar que esta función de evaluación, sus mínimos locales correponde a coloraciones propias factibles.

2.2.3. Actualización de la solución actual

Existen dos manera de actualizar una solución. Una de ellas es explorar todo el vecindario de la solución actual y reemplazarla por la «solución vecina» que posea el menor valor en la función definida en la sección anterior. Sin embargo, una de las dificultades de tal estrategía es el consumo de recursos, especialmente tiempo computacional por la exploración exhaustiva del vecindario, por lo que en el presente trabajo se ha considerado reemplazar la solución actual por la primera solución vecina que mejore el costo de la coloración actual.

Es importante destacar que la exploración se la realiza considerando coloraciones factibles y no factibles y que esta etapa se la ejecuta durante un tiempo máximo t_{max}.

A continuación se muestra el seudocódigo de la fase de búsqueda local

Algoritmo 5 Búsqueda Local

1: **procedure** BÚSQUEDALOCAL(*solución*) inicial $\leftarrow f(solución)$
2: **while** $Time\_max$ no se cumpla **do** Determine una solución $s \in N(solución)$ por medio del reemplazo del color de un vértice escogido aleatoriamente por un nuevo color.
3: **if** $f(s)$ <inicial **then**
solución $\leftarrow s$
inicial $\leftarrow f(s)$
4: **end if**
5: **end whilereturn** *solución*
6: **end procedure**

2.3. Calibración

Uno de los aspectos a analizar en la presente sección es la estrategia a seguir con respecto al tamaño de la lista RCL. Para tal efecto, se ejecutó el algoritmo desarrollado con algunas instancias del problema de coloración de grafos [1] durante 10 iteraciones cada una de ellas con una duración de 60 segundos. La implementación del algoritmo se la realizó en Mathematica 8.0.4.0® el cual «es un software especializado en Matemáticas desarrollado por Wolfram Research que integra diversas funcionalidades numéricas de cualquier precisión, capacidades simbólicas o de visualización, así como también sirve como lenguaje de programación para propósitos generales»[2].

En la tabla 2.1 se muestra un resumen de los resultados obtenidos al ejecutar el algoritmo en tres instancias específicas. En la primera columna se observa el nombre de la instancia, en la segunda columna el número de colores óptimo para realizar una coloración propia del grafo dado. En la tercera columna se muestran los porcentajes de aciertos con respecto al óptimo tanto de la estratégia con longitud de lista fija (F) y

[1] http://mat.gsia.cmu.edu/COLOR/instances.html#XXLEI
[2] http://www.wolfram.com/mathematica/features/

longitud variable (V) y por último los tiempos promedios en la obtención del mínimo número de colores requeridos, obtenidos en la ejecución de cada instancia.

		Resultados			
Instancia	# óptimo de colores	% de aciertos		tiempos(seg.)[a]	
		F	V	F	V
myciel5.col	6	100	100	3.852	3.392
myciel6.col	7	60	70	30.157	31.727
myciel7.col	8	0	0	49.600	46.425

Tabla 2.1: Calibración: Análisis de Resultados

[a]Tiempo promedio que toma en alcanzar el mínimo durante todas las iteraciones

En las figuras 2.1, 2.2 y 2.3, se realiza una comparación gráfica de los histogramas de frecuencias de las soluciones obtenidas al ejecutar el algoritmo con cada una de las instancias escogidas para el efecto, ya sea considerando la longitud de la lista RCL fija o variable.

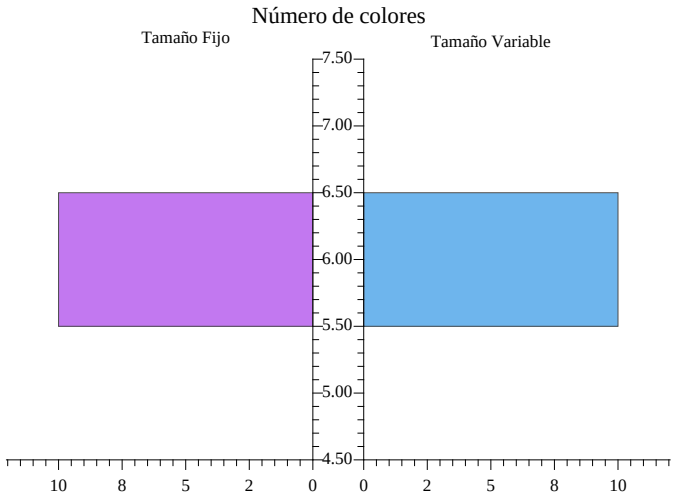

Figura 2.1: Histograma de frecuencia de la soluciones para la instancia myciel5.col. Número de colores óptimo:6

Se puede observar que en la figura 2.1, en ambos casos, en un 100 % de las iteraciones se alcanzó el óptimo global al ejecutar el algoritmo. Adicionalmente, de acuerdo a la

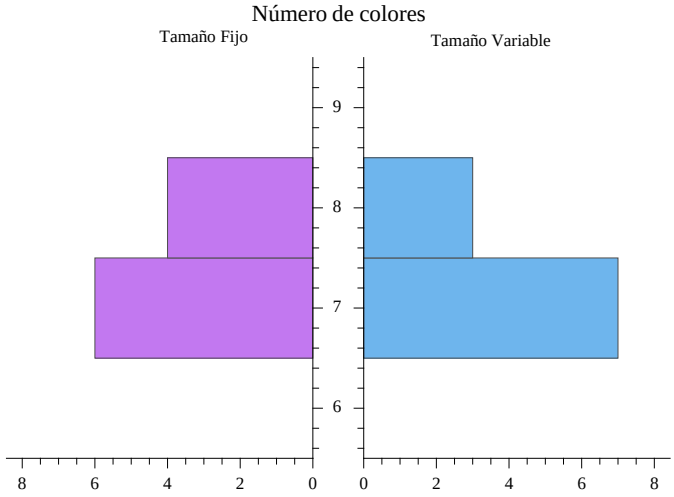

Figura 2.2: Histograma de frecuencia de la soluciones para la instancia myciel6.col. Número de colores óptimo:7

tabla 2.1 los tiempos promedios en que se alcanza este óptimo son similares, ya sea si se escoge una lista de longitud fija o variable. En la figura 2.2 se observa que los porcentajes de aciertos del algoritmo con el óptimo son similares, así como los tiempos promedios en alcanzar el mínimo global. Mas, en la figura 2.3 se observa que nunca se alcanza el óptimo global, sin embargo los resultados obtenidos considerando una lista variable son mejores que los obtenidos en la lista de longitud fija por cuanto se acercan al óptimo. Por lo tanto, y considerando que los tiempos promedios son similares en todas las instancias ejecutadas, se ha considerado la longitud de la lista RCL como variable, es decir, la lista RCL estará definida como en la ecuación 2.1

2.4. Resultados computacionales

Luego de la calibración de los parámetros del algoritmo GRASP desarrollado en la sección anterior, se ejecutó el mismo con diversas instancias en una PC con procesador

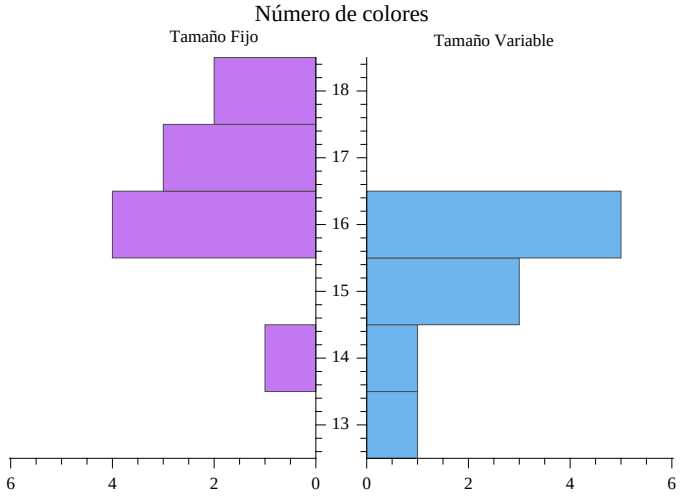

Figura 2.3: Histograma de frecuencia de la soluciones para la instancia myciel7.col. Número de colores óptimo:8

INTEL CORE i3 3.07 GHz de 64 bits, durante un máximo de MAX_ITER, cada una de ellas ejecutadas durante un tiempo t_max, cuyos resultados se muestran en la tabla 2.2.

Instancia	# óptimo de colores	# de nodos	# de arcos	Parámetros		# de colores
				MAX_ITER	$t\_max(seg)$	
queen8_8.col	9	64	728	400	45	10
queen9_9.col	10	81	2112	280	90	12
queen11_11.col	11	121	3960	255	120	14
myciel3.col	4	11	20	320	10	4
myciel4.col	5	23	71	320	10	5
myciel5.col	6	47	236	320	10	6
myciel6.col	7	95	755	320	10	7
myciel7.col	8	191	2360	320	10	9

Tabla 2.2: Resultados de la ejecución del algoritmo para diversas instancias

En las figuras 2.4,2.5 y 2.6 se muestra la tendencia del número de colores requeridos para una coloración propia en cada instancia utilizada en función del tiempo. Como se puede observar, el algoritmo es eficiente en la búsqueda de soluciones de buena calidad, especialmente para instancias de pequeña escala.

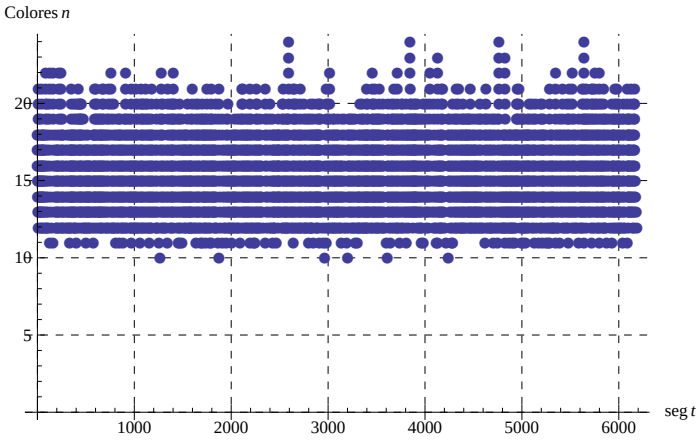

Figura 2.4: Gráfica de la evolución del número de colores requeridos en la instancia queen8_8.col

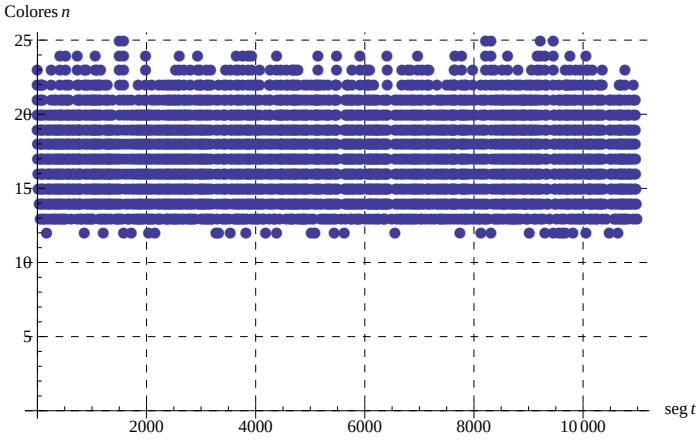

Figura 2.5: Gráfica de la evolución del número de colores requeridos en la instancia queen9_9.col

A continuación, en las figuras 2.7, 2.8 y 2.9 se muestra la tendencia del número de colores requeridos para una coloración propia, pero sólo en el mínimo valor encontrado luego de ejecutar el número máximo de iteraciones. Como se puede observar, luego que se ha determinado una solución factible inicial, el tiempo que toma en alcanzar el mínimo local es razonable en comparación con los que se toman al aplicar un modelo que los resuelva en forma exacta.

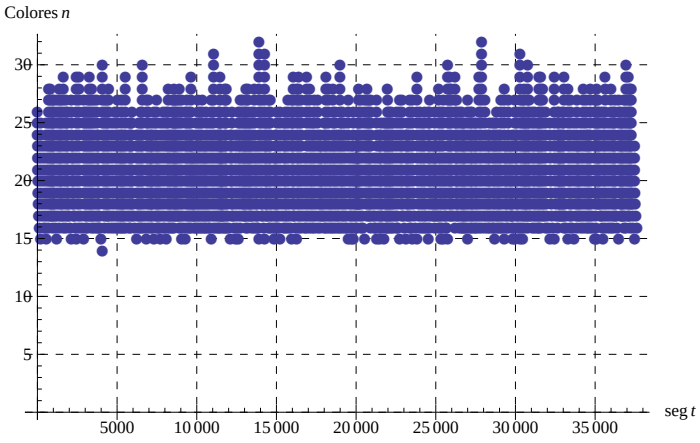

Figura 2.6: Gráfica de la evolución del número de colores requeridos en la instancia queen11_11.col

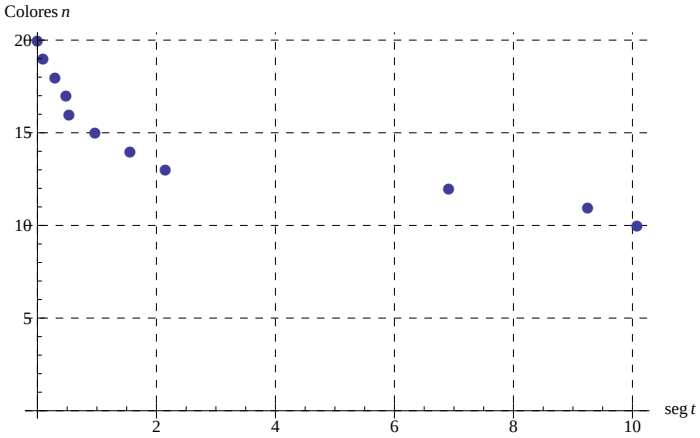

Figura 2.7: Evolución del número de colores requeridos en la instancia queen8_8.col

Otra forma de validar los resultados que se obtienen al ejecutar el algoritmo propuesto para diversas instancias, es por medio de la comparación de los mismos con los que se obtienen al aplicar una función específica de la libreria de Mathematica 8.0.4.0® denominada «VertexColoring», la misma que consiste en la implementación del algoritmo de Brélaz, D. [12]. En la tabla 2.3 se muestra los resultados al ejecutar diversas instancias tanto con la heurística propuesta como con el algoritmo de Brélaz

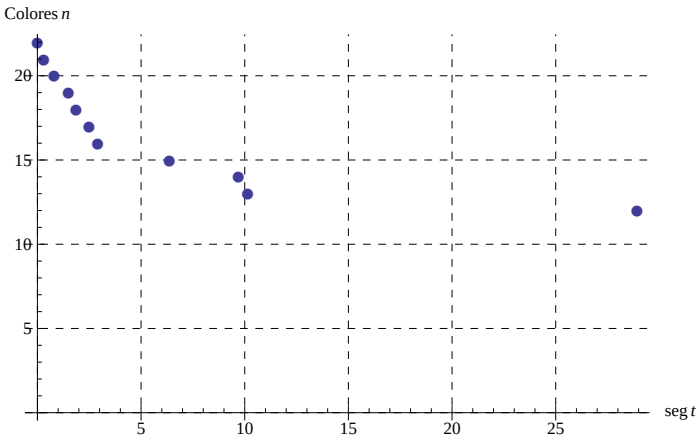

Figura 2.8: Evolución del número de colores requeridos en la instancia queen9_9.col

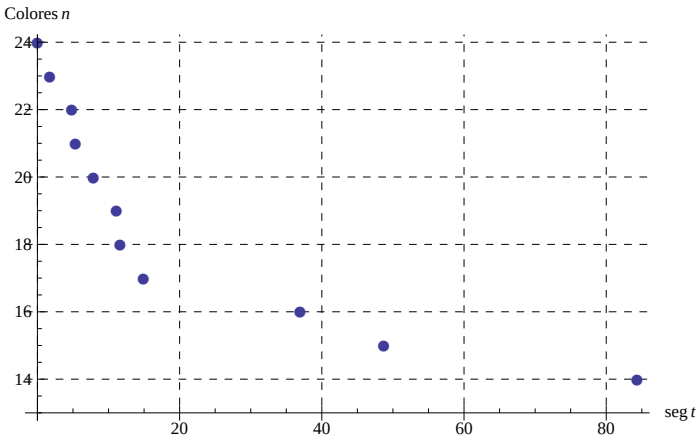

Figura 2.9: Evolución del número de colores requeridos en la instancia queen11_11.col

Instancia	Heurística Propuesta	Algoritmo Brelaz
queen8_8.col	10	15
queen9_9.col	12	15
queen11_11.col	14	16
myciel3.col	4	4
myciel4.col	5	5
myciel5.col	6	6
myciel6.col	7	7
myciel7.col	9	8

Tabla 2.3: Comparación entre heurística propuesta y el algoritmo de Brélaz

Capítulo 3

Implementación de la heurística propuesta al problema de calendarización de exámenes

Una de las aplicaciones de los algoritmos diseñados para el Problema de Coloración de Grafos es en la confección de horarios de exámenes en una institución educativa. En este contexto, se ha considerado aplicar el algoritmo propuesto en el capítulo anterior en la planificación de los horarios de exámenes en un semestre específico de estudios, de una carrera específica de una institución educativa de nuestro medio, por lo que se hace necesario establecer diversas consideraciones operativas propias de la organización que son importantes en el diseño de horarios factibles.

3.1. Restricciones

Diversas restricciones operativas se presentan en el momento de elaborar cualquier planificación. Específicamente, en la confección de los horarios de exámenes en la que se pretende aplicar el algoritmo propuesto, se han considerado las siguientes restricciones.

i) Restricciones fuertes.

a) Uno de los mayores problemas que tiene el estudiantado de cualquier institución educativa es la que exámenes considerados fuertes se planifiquen el mismo día. Por ello, y como una forma de impulsar su desempeño académico, se ha considerado que los exámenes de las materias fuertes ubicadas en el mismo nivel se lo recepten en días diferentes.

b) De igual manera, se ha considerado un máximo de 30 aulas disponibles en un día, por lo que en cada día no se puede programar materias que en conjunto superen el número máximo de aulas disponibles.

II) Restricciones suaves.

a) Se ha considerado el hecho de que en cada semestre planificado se penalice la posibilidad de evaluar más dos exámenes del mismo nivel.

Con base en lo establecido anteriormente, es necesario conocer las materias y el número de paralelos aperturados por cada materia. En la tabla 3.1 se muestran las materias por nivel. De igual manera, entre paréntesis se muestra el número de paralelos por cada materia.

De igual manera, en la tabla 3.2 en cada fila se muestran las materias en la que las evaluaciones de la primera columna deben estar planificadas en días distintos a la restante por su nivel de complejidad.

Nivel	Materias				
1	MatI914(14)	MatI2496(1)	MatI039(1)		
2	MatI985(11)	MatI640(13)	MatI910(1)	MatI749(10)	MatI236(2)
3	MatI185(6)	MatI623(2)	MatI966(10)	MatI444(1)	MatI947(11)
4	MatI127(1)	MatI415(1)	MatI166(8)		
5	MatI402(1)	MatI496(1)	MatI458(1)		
6	MatF177(1)	MatI591(1)	MatF115(1)	MatF144(1)	MatI477(1)
7	MatI514(3)				
8	MatI326(2)	MatI484(2)			
9	MatI548(1)	MatI913(1)	MatI529(1)	MatI618(1)	

Tabla 3.1: Materias impartidas por niveles

Materia	Materias que deben evaluarse en días diferentes a la de la columna 1					
MatI548	MatI185	MatI947				
MatI127	MatI966	MatI166	MatI947			
MatF115	MatI591	MatI402	MatI477			
MatI913	MatI548	MatI185				
MatI591	MatF115	MatI477				
MatI185	MatI548	MatI913	MatI947			
MatI402	MatF115					
MatI749	MatI910	MatI640	MatI914	MatI985		
MatI477	MatF115	MatI591	MatI947			
MatI966	MatI127	MatI640	MatI947			
MatI166	MatI127	MatI947				
MatI640	MatI910	MatI966	MatI914	MatI985		
MatI914	MatI910	MatI749	MatI640			
MatI985	MatI910	MatI749	MatI640			
MatI947	MatI548	MatI127	MatI185	MatI477	MatI966	MatI166

Tabla 3.2: Materias que no deben evaluarse en el mismo día

3.1.1. Formulación matemática del Problema de Calendarización de Exámenes aplicado en una institución educativa

A continuación se presenta una variante del modelo de programación no lineal, definido en capítulo anteriores, que permite encontrar una solución óptima al problema

de calendarización de exámenes aplicado específicamente en una institución académica, considerando la mínima penalización por violación de las restricciones suaves.

Conjuntos y parámetros

I) Exámenes:

a) E: Conjunto de exámenes.

b) se_i: Número de estudiantes registrados en el examen $i \in E$

II) Estudiantes:

a) S: Conjunto de estudiantes

b) $t_{is} = \begin{cases} 1, & \text{si el estudiante } s \text{ debe rendir el examen } i \\ 0, & \text{si no} \end{cases}$

III) Aulas

a) R conjunto de aulas

b) s_r: Capacidad del aula $r \in R$

IV) períodos

a) P : Conjunto de períodos

b) $y_{pq} = \begin{cases} 1, & \text{si los periodos } p \text{ y } q \text{ se encuentran en el mismo día} \\ 0, & \text{si no} \end{cases}$

V) Penalizaciones

a) w^{2R}: penalización por exámenes consecutivos.

b) w^{2D}: penalización por exámenes administrados en el mismo día.

Variables

I) $xp_{ip} = \begin{cases} 1, & \text{si el examen } i \text{ debe programarse en el período } p \\ 0, & \text{si no} \end{cases}$

II) $xr_{ir} = \begin{cases} 1, & \text{si el examen } i \text{ debe programarse en el aula } r \\ 0, & \text{si no} \end{cases}$

III) $C2R_s$ Variable que indica el número de ocasiones que el estudiante s tiene dos exámenes consecutivos en un día.

IV) $C2D_s$ Variable que indica el número de ocasiones que el estudiante s tiene dos exámenes planificados en un día.

Formulación Matemática

$$\min \sum_{s \in S} w^{2R} C2R_s + w^{2D} C2D_s \tag{3.1}$$

$$\sum_{r \in R} xr_{ir} = 1, \ \ \forall i \in E \tag{3.2}$$

$$\sum_{p \in P} xp_{ip} = 1, \ \ \forall i \in E \tag{3.3}$$

$$se_i xp_{ip} xr_{ir} \leq s_r, \ \ \forall p \in P, \ \ \forall i \in E, \ \ \forall r \in R, \tag{3.4}$$

$$\sum_{i \in E} t_{is} xp_{ip} \leq 1, \ \ \forall p \in P, \ \ \forall s \in S \tag{3.5}$$

$$\sum_{p=4*dia-3}^{4*dia} xp_{ip} + \sum_{p=4*dia-3}^{4*dia} xp_{jp} \leq 1, \ \ \forall dia = 1,2,3,4,5 \tag{3.6}$$

$\forall i, j$ son materias de la misma fila de la tabla 3.2

$$C2R_s = \sum_{\substack{i,\, j \,\in\, E \\ i \neq j}} \sum_{\substack{p,\, q \,\in\, P, \\ q \,=\, p+1, \\ y_{pq} \,=\, 1}} t_{is} t_{js} xp_{ip} xp_{jq} \tag{3.7}$$

$$C2D_s = \sum_{\substack{i,\, j \,\in\, E \\ i \neq j}} \sum_{\substack{p,\, q \,\in\, P \\ q \,>\, p+1 \\ y_{pq} \,=\, 1}} t_{is} t_{js} xp_{ip} xp_{jq} \tag{3.8}$$

$$xp_{ip}, xr_{ir} \in \{0,1\} \tag{3.9}$$

$$C2R_s, C2D_s \in Z, C2R_s \geq 0, C2D_s \geq 0 \forall s \in S \tag{3.10}$$

La ecuación (3.1) establece la función objetivo la cual es una combinación lineal del número de estudiantes que tienen exámenes consecutivos, así como exámenes que se toman en el mismo día. La restricción (3.2) establece que cada examen sea administrado en alguna aula. La restricción (3.3) establece que cada examen sea administrado en algún

período. La restricción (3.4) garantiza que ningún examen sea administrado en un aula con capacidad menor al número de alumnos que deben rendir la prueba. La restricción (3.5) garantiza que cada estudiante en cualquier período rinda a lo mucho un examen. La restricción (3.6) garantiza que en cada día, a lo mucho un examen considerado difícil sea administrado. Las ecuaciones (3.7) y (3.8) permiten determinar el número de estudiantes que tienen planificado más de un examen en el mismo día, así como los que rinden dos exámenes consecutivos. Por último las restricciones (3.9) y (3.10) se refieren al caracter de las variables declaradas en el modelo.

3.2. Resultados Computacionales

A continuación se exponen los resultados obtenidos de la implementación de la heurística propuesta en el problema de calendarización de exámenes aplicado a una carrera de pregrado de una institucion académica.

	Lunes	Martes	Miércoles	Jueves	Viernes
8:30 10:30		MATF039			
11:00 13:00					
13:30 15:30			MATI914		MATI2496
16:00 18:00					

Tabla 3.3: Horario de exámenes de materias del primer nivel

	Lunes	Martes	Miércoles	Jueves	Viernes
8:30 10:30		MATI640		MATI910	
11:00 13:00					
13:30 15:30	MATI749		MATI985		MATI236
16:00 18:00					

Tabla 3.4: Horario de exámenes de materias del segundo nivel

	Lunes	Martes	Miércoles	Jueves	Viernes
8:30 10:30	MATI947	MATI185		MATI966	
11:00 13:00					
13:30 15:30			MATI623		MATI444
16:00 18:00					

Tabla 3.5: Horario de exámenes de materias del tercer nivel

	Lunes	Martes	Miércoles	Jueves	Viernes
8:30 10:30				MATI166	
11:00 13:00		MATI415			
13:30 15:30			MATI127		
16:00 18:00					

Tabla 3.6: Horario de exámenes de materias del cuarto nivel

	Lunes	Martes	Miércoles	Jueves	Viernes
8:30 10:30				MATI496	
11:00 13:00		MATI402			
13:30 15:30					MATI458
16:00 18:00					

Tabla 3.7: Horario de exámenes de materias del quinto nivel

	Lunes	Martes	Miércoles	Jueves	Viernes
8:30 10:30		MATF177		MATI477	
11:00 13:00		MATI591			
13:30 15:30	MATF115				MATF144
16:00 18:00					

Tabla 3.8: Horario de exámenes de materias del sexto nivel

	Lunes	Martes	Miércoles	Jueves	Viernes
8:30 10:30				MATI514	
11:00 13:00					
13:30 15:30					
16:00 18:00					

Tabla 3.9: Horario de exámenes de materias del séptimo nivel

	Lunes	Martes	Miércoles	Jueves	Viernes
8:30 10:30				MATI484	
11:00 13:00					
13:30 15:30	MATI326				
16:00 18:00					

Tabla 3.10: Horario de exámenes de materias del octavo nivel

	Lunes	Martes	Miércoles	Jueves	Viernes
8:30 10:30	MATI618			MATI913	
11:00 13:00					
13:30 15:30	MATI529		MATI548		
16:00 18:00					

Tabla 3.11: Horario de exámenes de materias del noveno nivel

Como se puede observar en cada una de las tablas mostradas anteriormente, se cumplen cada una de las restricciones fuertes establecidas anteriormente.

Capítulo 4

Conclusiones y recomendaciones

El presente trabajo se lo realizó en dos etapas. La primera consistió en el diseño de un algoritmo GRASP para el problema de coloración de grafos. Como se puede observar en la tabla 2.2, los resultados son satisfactorios en la búsqueda de soluciones factibles en un tiempo computacional razonable. Sin duda alguna, esto evidencia la utilidad de aplicar heurísticas a problemas combinatorios de complejidad computacional elevada en comparación a si se los resuelve por métodos exáctos.

Sin embargo es notorio la pérdida de eficiencia en instancias de gran tamaño. Diversas podrían ser las causas para tal hecho. En efecto, en las gráficas 2.4, 2.5, 2.6 se observa la rápida convergencia en el proceso de búsqueda local; sin embargo, es evidente que la solución inicial no es de buena calidad por cuanto existe una gran variabilidad con respecto al valor óptimo. En futuras investigaciones, a priori, se podría trabajar en el proceso de construcción de la solución inicial. Una de las estrategias a considerar es que en la exploración del espacio de soluciones factibles, no se utilizan estructuras de memoria a corto y largo plazo por lo que se recomienda en futuros trabajos incorporarlos.

Por otro lado, dado que el problema de coloración de grafos puede ser modelizado como uno de particionamiento de conjuntos, se podría estudiar otras estrategias enfocadas en esta temática. Una de ellas es la de generación de columnas.

La segunda etapa básicamente consistió en la aplicación del algoritmo diseñado en el problema de calendarización de exámenes. En este caso, sólo se ha considerado como restricciones fuertes el hecho de que materias de complejidad elevada sean evaluadas en días distintos, así como el número de paralelos evaluados en un día específico no supere un límite. Sin embargo se debe considerar restricciones adicionales como el hecho de que existen materias que los estudiantes dejan rezagadas período a período y que en definitiva afecta la confección de calendarios factibles.

Apéndices

Apéndices A

Código Fuente del algoritmo propuesto

```
saturaciondegree[x_, y_] :=

  If[1 == 1, satur = Table[If[Part[y, i] == 0, {i, 0}, {i, -10}], {i, Length[y]}];

   For[i = 1, i <= Length[y],

      If[Part[y, i] == 0,

         temporal = {};

         For[j = 1, j <= Length[y],

           If[Part[y, i] != Part[y, j]  &&  Part[x, i, j] == 1 && Part[y, j] != 0,

             temporal = AppendTo[temporal, {Part[y, j]}]

             ];

         j++];

         temporal = Union[temporal, {}];

         satur = ReplacePart[satur, i -> {i, Length[temporal]}]

        ];

      i++]
```

```
    ];
objetivo2[xx_, yy_] := If[1 == 1, mejor = Table[0, {i, Max[xx]}];
      particion = Table[Flatten[Position[xx, i]], {i, Max[xx]}];
    For[kk = 1, kk <= Length[particion],
     For[kkk = 1, kkk <= Length[Part[particion, kk]] - 1,
      For[kkkk = kkk, kkkk <= Length[Part[particion, kk]],
       If[Position[Part[yy, Part[particion, kk, kkk]],
         Part[particion, kk, kkkk]] != {},
        mejor = ReplacePart[mejor, kk -> Part[mejor, kk] + 1 ]
        ]
            ; kkkk++]
        ; kkk++]
         ; kk++ ]];
objetivo[x_] := If[1 == 1,lista = Table[0, {i, Max[x]}];
      For[i = 1, i <= Length[x],
         lista=ReplacePart[lista, Part[x, i] -> Part[lista, Part[x, i]] + 1];
        i++];
     objetivo2[x, relacion23]; cuadrado = 0;
   cuadrado = \!\(
\*UnderoverscriptBox[\(\[Sum]\), \(i = 1\), \(Length[
       lista]\)]\((2*Part[lista, i]*Part[mejor, i] -
\*SuperscriptBox[\((Part[lista, i])\), \(2\)])\)\);
```

```
   ];
data = Import["D:\\ESPOL\\tesiss\\data23.txt", "table"];
data = Union[data, Map[Reverse, data]];
incidencia = Table[0, {i, Max[data]}, {j, Max[data]}];
incidencia = ReplacePart[incidencia, data -> 1];
parcial = Total[incidencia, {2}];
grado = Table[{i, Part[parcial, i]}, {i, Length[parcial]}];
vertices = Sort[grado, #1[[2]] < #2[[2]] &];
candi = {};
listasi = {};
For[jjj = 1, jjj <= 400,
dosi = RandomInteger[{1000}];
SeedRandom[dosi + 1];
alpha = RandomReal[];
maximo = Max[Take[vertices, Length[vertices], -1]];
minimo = Min[Take[vertices, Length[vertices], -1]];
par=minimo + alpha*(maximo - minimo);
escoger=Table[If[ Part[vertices, i, 2] >= par, Part[vertices, i, 1 ],0]
,{i, Length[vertices]}];
escoger=Delete[escoger, Position[escoger, 0]];
bandera=RandomInteger[{1, Length[escoger]}];
color=Table[0, {i, Length[grado]}];
```

```
color=ReplacePart[color, Part[escoger, bandera] -> 1];

relacion=Table[If[Part[incidencia, i, j] == 1, {j}, {}], {i,Length[incidencia]}

, {j, Length[incidencia]}];

relacion=Delete[relacion, Position[relacion, {}]];

relacion23=Table[If[Part[incidencia, i, j] == 1, j, {}], {i,Length[incidencia]}

, {j, Length[incidencia]}];

 relacion23 = Delete[relacion23, Position[relacion23, {}]];

 While[Position[color, 0] != {}, noc = {0};

saturaciondegree[incidencia, color];

satur = Sort[satur, #1[[2]] < #2[[2]] &];

dosi = RandomInteger[{1000}];

SeedRandom[dosi + 1];

alpha = RandomReal[];

maximo = Max[Take[satur, Length[satur], -1]];

minimo = Min[Take[satur, Length[satur], -1]];

par = minimo + alpha*(maximo - minimo);

escoger=Table[If[ Part[satur, i, 2] >= par, Part[satur, i, 1 ], 0]

,{i,Length[satur]}];

escoger = Delete[escoger, Position[escoger, 0]];

bandera = RandomInteger[{1, Length[escoger]}];

dos = Part[escoger, bandera];

uno = {dos};
```

```
If[Length[uno] == 1 &&  Part[color, Part[uno, 1]] == 0,

For[i = 1, i <= Length[Part[relacion, Part[uno, 1]]],

noc = AppendTo[noc,Part[color, Part[relacion, Part [uno, 1], i, 1]]] ;

i++];

enn = RandomInteger[{1, Max[color] + 1}];

While[Position[noc, enn] != {},

enn = RandomInteger[{1, Max[color] + 1}]];

color = ReplacePart[color, Part[uno, 1] -> enn];]

   ];

objetivo[color]; esi = cuadrado;

candidato = Length[DeleteDuplicates[color]];

cand = {};

start = SessionTime[];

cand = AppendTo[cand, {SessionTime[] - start, candidato}];

Timing[TimeConstrained[Do[

posicion = RandomInteger[{1, Length[color]}];

escojo = Part[color, posicion];

color23 = color;

at = RandomInteger[{1, Max[color] + 1}] ;

While[ at == escojo ,

at = RandomInteger[{1, Max[color] + 1}]

];
```

```
color23 = ReplacePart[color23, posicion -> at]; objetivo[color23];

amigo = Length[DeleteDuplicates[color23]];

If[Total[mejor] == 0  &&  amigo < candidato, candidato = amigo;

vb = color23;

cand = AppendTo[cand, {SessionTime[] - start, amigo}]

];

    If[esi > cuadrado, color = color23; esi = cuadrado]

    , {1000000000}

]

   , 45]]; candi = AppendTo[candi, cand]; listasi = AppendTo[listasi, vb]

; jjj++]
```

Apéndices B

Glosario

Partición de un conjunto A: Dado un conjunto de índices I, y $\forall i \in I \ \ \phi \neq A_i \subseteq A$, entonces $\{A_i\}_{i \in I}$ es una partición de A si $A = \bigcup_{i \in I} A_i$ y $\forall i, j \in I, i \neq j \ A_i \cap A_j = \phi$

Grafo dirigido: O digrafo, es el par ordenado (V, E), donde V es un conjunto no vacío denominado Vértices o Nodos, y $E \subseteq V \times V$ denominado conjunto de aristas, donde cada arista es un par ordenado de vértices.

Grafo no dirigido: Es el par ordenado (V, E), donde V es un conjunto no vacío denominado Vértices o Nodos, y $E \subseteq V \times V$ denominado conjunto de arcos, donde cada arco es un par no ordenado de vértices.

Grafo simple: Es un grafo $G = (V, E)$, que admite máximo una arista(o arco) entre cada par de vértices de V en el caso de grafos dirigidos (no dirigidos).

Coloración propia del grafo $G = (V, E)$: Es una función $f : V \to [1, k]$ tal que $\forall (a, b) \in E$ se cumple que $f(a) \neq f(b)$

Número cromático de un grafo $G = (V, E)$: Es el mínimo número k de colores diferentes que da la posibilidad de realizar una coloración propia del grafo $G = (V, E)$

Nodo adyacente: Sea $G = (V, E)$ un grafo. Se dice que v es un nodo adyacente a w si existe un arco $(w, v) \in E$.

Metaheurísticas: Son métodos de solución que integran procedimientos de búsqueda local y estrategias de alto nivel que permiten escapar de óptimos locales y realizar una «búsqueda robusta» en el espacio de soluciones.

Coloración parcial de un grafo $G = (V, E)$: Es una relación $f : V \to [1, k]$ tal que $domf \neq V$ y $\forall a, b \in domf$ se cumple que $f(a) \neq f(b)$

Grado de v: Sea $G = (V, E)$ un grafo no dirigido. Para cada vértice $v \in V$, el grado de v denotado por $deg(v)$ es el número de arcos que son incidentes con v.

Grado de saturación de v: Sea G un grafo simple y C una coloración parcial de G. Se define al grado de saturación de un vértice como el número de diferentes colores asignados a sus vértices adyacentes.

Búsqueda tabú: es un procedimiento metaheurístico para guiar un algorítmo heurístico de búsqueda local en la exploración del espacio de soluciones más allá de la simple optimalidad local.

Algoritmo genético: Es un proceso metaheurístico basado en la teoría de la evolución humana que partiendo de una población inicial, explora el espacio de soluciones por medio de reglas de selección, mutación y de cruce de soluciones factibles de un problema de optimización.

Recocido simulado: Es un proceso metaheurístico basado en procesos físicos y químicos de los materiales que permite encontrar soluciones factibles de buena calidad.

Vecindario de una solución v: es el conjunto formado por todas las soluciones w que se obtienen al realizar un «movimiento» o «cambio» en v.

Bibliografía

[1] McCollum B. & McMullan P. (2007) «*The Second International Timetabling Competition: Examination Timetabling Track*». Queen's University Belfast.

[2] Du D. & Panos M. (2005), «*Handbook of combinatorial optimization, Volumen B*», Springer.

[3] Lu Z. & Hao J. (2009) «*A memetic algorithm for graph coloring*», European Journal of Operations Research, ELSEVIER.

[4] Cédric A. & Hertz A. & Zufferey N. (2003), «*A variable neighborhood search for graph coloring*». European Journal of Operational Research, ELSEVIER.

[5] Hertz A. & Werra D. (1987). «*Using Tabu Search Techniques for Graph Coloring*». Computing by Springer-Verlag.

[6] Lucet C. & Mendes F., Moukrim A. (2004).«*An Exact method for graph coloring*». Computer & Operations Research, ELSEVIER.

[7] Méndez I. & Zabala P.. (2007) «*A cutting plane algorithm for graph coloring*». Discrete Applied Mathematics. ELSEVIER.

[8] Eiben A. & Van der Hauw J. & Van Hennert J. «*Graph Coloring with Adaptive Evolutionary Algorithms*». Journal of Heuristics, volume 4:1.

[9] Bodlaender H. & Kratsch D. (2006) «*An exact algorithm for graph coloring with polynomial memory*». Utrecht University.

[10] Johnson D. & Aragón C. & Mcgeoch L. & Schevon C. (1990) «*Optimization by simulated annealing: An Experimental Evaluation;Part II, Graph Coloring and Number Partitiong*», Operations Research Society of America.

[11] Werra D.(1990) «*Heuristics for Graph Coloring, Computational Graph Theory*», Comput. Suppl. 7, Springer, Vienna, 191-208.

[12] Brélaz, D. (1979) «*New methods to color the vertices of a graph*». Communications of the Assoc. of Comput. Machinery 22, 251-256.

[13] Resende M. & Ribeiro C. (2002) «*Greedy Randomized Adaptive Search Procedures*», AT&T Labs Research Technical Report. Hanbook in Metaheuristic, Fred Glover.

[14] Glover F. (2003) «*HandBook of Metaheuristics*»,International Series in Operations Research & Management Science.

[15] Talbi E. (2009) «*Metaheristics from design to implementaction*», John Wiley & Sons, Inc., Hoboken, New Jersey

[16] R. Grimaldi (2004) «*Discrete and Combinatorial Mathematics*», Pearson, Fifth Edition.