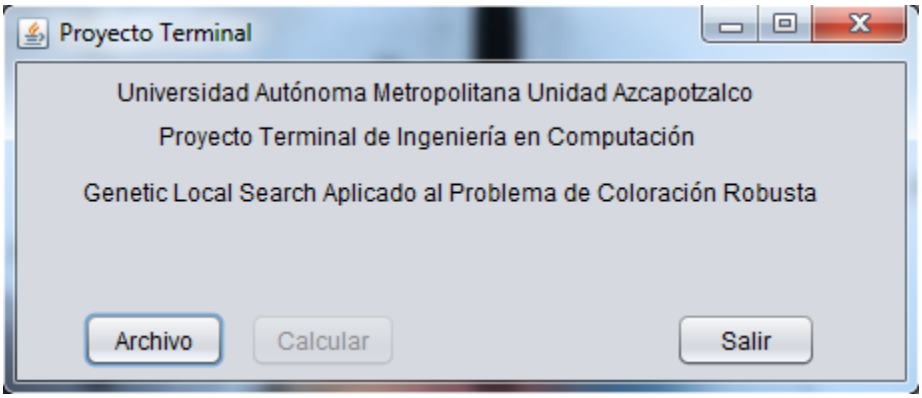
Manual de usuario para la aplicación de Proyecto Terminal: GLS aplicado al problema de coloración robusta.

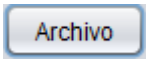
La aplicación lleva como nombre GLSApp.jar; para poder hacer funcionar la aplicación se necesita descargar la máquina virtual de Java desde la página de Oracle.

Posteriormente bastará con dar doble clic en la aplicación para ejecutarla:

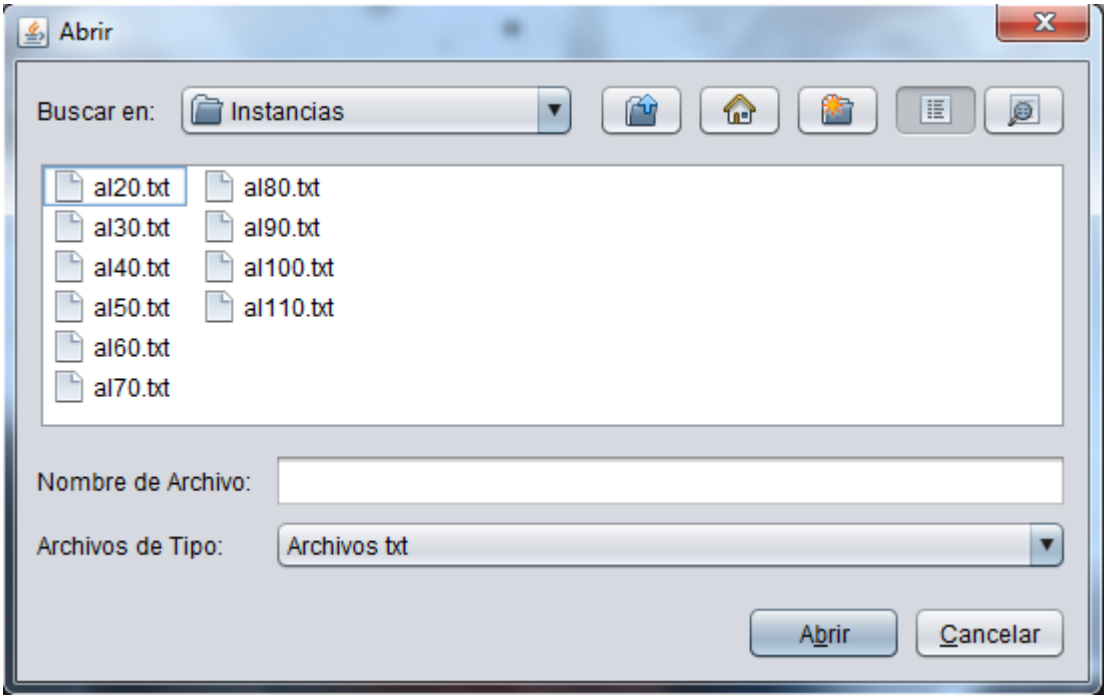
Al ejecutarse se mostrará la siguiente Pantalla:



En dicha pantalla existirán 3 botones, de los cuales sólo dos se encontrarán habilitados.



Permite al usuario seleccionar un archivo .txt que hace referencia a la instancia desplegando un cuadro de dialogo como sigue:

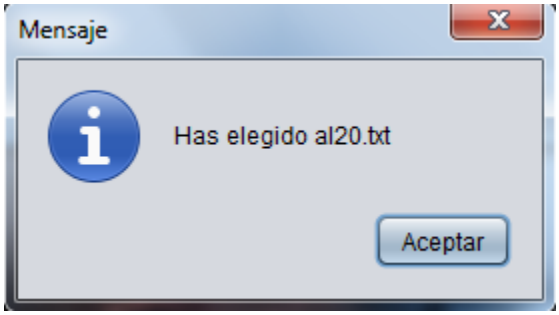


En esta etapa se podrá seleccionar un archivo que corresponda a un problema referente al PCR.

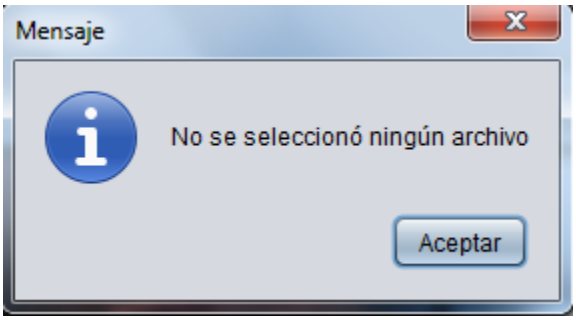
La estructura del archivo es como sigue:

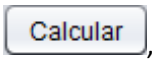
Se tratan de matrices cuadradas que indican la relación que existe entre los nodos. En cada elemento de la matriz se define la probabilidad de cambio en las aristas, esto es, que a cada elemento le corresponde una rigidez dada. En algunos casos la rigidez es 1, lo cual significa tiene un conflicto, de otra manera se trata de una rigidez asociada a ese par de grafos (fila contra columna).

Una vez seleccionado el archivo se debe dar clic en “abrir” y aparecerá la siguiente ventana:



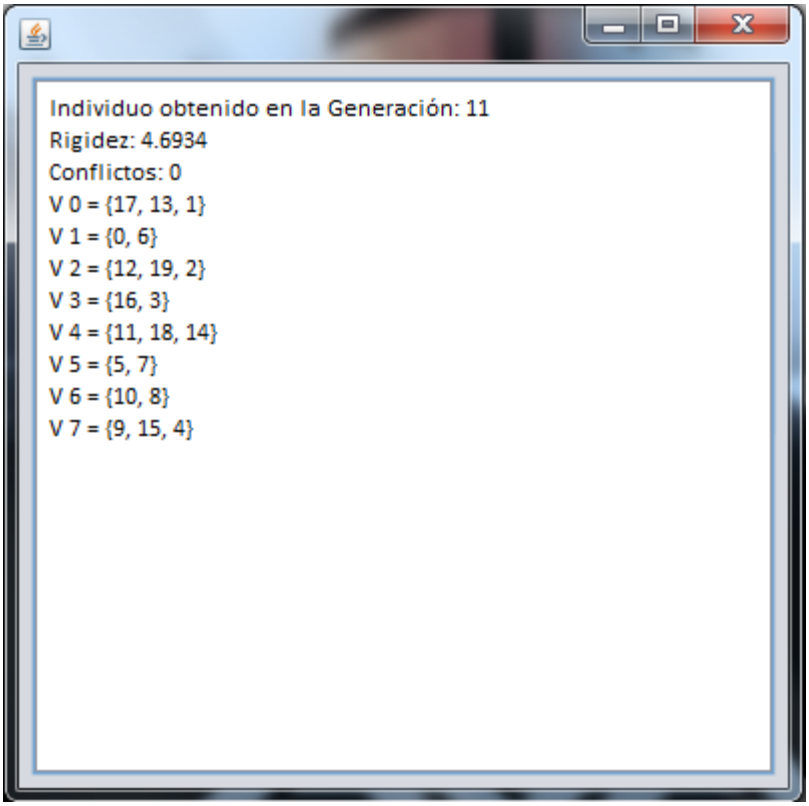
De lo contrario se abrirá la siguiente ventana al dar clic en cancelar.



Una vez seleccionado el archivo se habilitará el botón 

, que permitirá utilizar la heurística de GLS sobre el problema seleccionado.

Una vez que se haya dado clic aparecerá la siguiente ventana:



Esta ventana mostrará el resultado obtenido por el algoritmo:

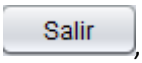
El número de la generación donde encontró el mejor individuo.

La rigidez de dicho individuo.

Los conflictos que presentó la solución.

Y una representación del grafo en forma de partición como se detalla en el reporte del proyecto terminal.

Nota: Debido al funcionamiento del algoritmo esta ventana puede tardar en responder en instancias más grandes. Para dichas situaciones el usuario deberá esperar una aproximación al tiempo que se menciona en la sección de resultados del reporte.

Para Salir de la aplicación puede usar el botón 

, o el botón situado en la parte superior de la ventana principal de “Proyecto Terminal”.

 , Estos últimos se podrán ver de forma diferente según el sistema operativo en el que se corra la aplicación, ya que su diseño no depende de la máquina virtual de Java.

Universidad Autónoma Metropolitana Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Licenciatura en Ingeniería en Computación

Proyecto Terminal

Adaptación e implementación del algoritmo de búsqueda local genética para resolver el problema de coloración robusta.

Lázaro Adrián González Montoya, matrícula 205300075

Asesor: Antonin Ponsich, profesor del Departamento de Sistemas, Categoría Asociado D, Número económico 35009

Trimestre 12-O

Diciembre 2012

Índice.

1. Introducción.

El presente trabajo presenta una adaptación e implementación del algoritmo de búsqueda local genética (GLS por sus siglas en inglés) como solución al Problema de Coloración Robusta (PCR). Dicho problema, como se explica en secciones posteriores, requiere de métodos heurísticos para obtener una solución en un tiempo razonablesobre instancias medias o grandes.Sin embargo, esto deja un espacio abierto a investigaciones sobre técnicas quepermitan mejorar las soluciones del estado del arte.

Siendo así, ya existen diversos métodos (búsqueda dispersa, recocido simulado, algoritmos evolutivos, etc…) que han sido probados para el problema en mención. Sin embargo, existe una nueva tendencia sobre los métodos híbridos, la cual consiste en combinar técnicas, que en otros problemas de similar complejidad, han obtenido muy buenos resultados como se muestra en el trabajo de Galinier y Hao [1]. Dichos autores son citados en repetidas ocasiones ya el algoritmo memético que presentan para el problema de coloración clásica, es tomado como modelo en diversas partes para el presente trabajo.

En el marco de este proyecto, se determinaron e implementaron en primer lugar los procesos internos de la parte evolutiva del algoritmo híbrido. Posteriormente, se implementó un optimizador local de tipo“escalando-la-colina” (*hill-climber*). Finalmente, ambos elementos fueron combinadosy se implementó la hibridación entre ellos, resultando en el algoritmo de GLS adaptado al problema PCR.

Se realizaron entonces varios experimentos computacionales, divididos entre dos fases. La fase de cálculos preliminares sirvió para ajustar los parámetros de operación del algoritmo. En la segunda fase, se efectuaron corridas intensivas sobre el banco de instancias propuesto,para conseguir los resultados definitivos del algoritmo GLS implementado.

Estos resultados permiten evaluar el desempeño del algoritmo, en términos de eficiencia (tiempo de ejecución) y de eficacia (calidad de soluciones), comparable con el de otras técnicas heurísticas. Sin embargo, existen posibilidades de mejora significativa del algoritmo de GLS que podrán ser exploradas en trabajos futuros.

El presente reporte se organiza como descrito a continuación. En una segunda sección, se describe el problema tratado y se presentan los trabajos relacionados con el mismo tema. En la sección 3, se definen los diferentes elementos del diseño del algoritmo desarrollado en este proyecto, mientras que aspectos relacionados con su implementación se explican en la sección 4. Los experimentos numéricos realizados se describen en la sección 5 y la sección 6 propone conclusiones y algunas perspectivas de trabajo futuro.

2. Descripción del problema.

2.1.Formulación.

De manera sintética, la versión clásica del problema de coloración de gráficas (PCG) consiste, dado un grafo no orientado *G(V,E),* en asignar un color a cada vértice de forma que cualquier vértice adyacente no tenga el mismo color. Se busca entonces la asignación de mínima cardinalidad (i.e., la coloración que use el mínimo número colores); al número de colores de la solución óptima se le denota como número cromático.

De este problema, ya ampliamente estudiado en el área de Investigación de Operaciones, surge el problema de coloración robusta (PCR). Se considera en esta variante la posibilidad de agregar nuevas aristas (llamadas aristas complementarias) al grafo inicial[2]. De esta manera, el objetivo se concentra en minimizar la probabilidad que no permanezca válida la solución (la coloración) al ser agregadas dichas aristas.

Definición. Dados el grafo *G(V,E),* un número de colores válido *k* y la familia de penalizaciones (también llamadas rigideces) $\{p_{ij} \geq 0, (i,j) \in \overline{E}\}$ asignadas a las aristas complementarias, se minimiza el grado de rigidez total de la *k*-coloración c^k, igual a la suma de las penalizaciones de las aristas complementarias cuyos extremos están igualmente coloreados [2]:

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

Modelizaciones matemáticas del PCR han sido propuestas, conduciendo al uso de técnicas de programación entera [2],[3]. Sin embargo, se ha demostrado que el PCR es un problema NP-Duro[1], así que la forma más adecuada de encontrar buenas soluciones, en un tiempo razonable de cómputo, para instancias de tamaño medio y grande es utilizando técnicas heurísticas. Algunas de ellas, como algoritmos glotones, métodos de búsqueda localo algoritmos evolutivos, han sido introducidas en[4] y [5] por ejemplo.

2.2.Aplicaciones.

Como se ha mencionado anteriormente el PCR es una variación del problema clásico de coloración. Una aplicación clásica de los problemas de coloración de gráficas es la asignación de recursos compatibles. En su formulación clásica (PCG), es común que el objetivo sea determinar el mínimo número de recursos, de tal manera que a cada par de elementos a programar que no pueden compartir el mismo recurso se les asigne un color (es decir un recurso) diferente.

[1]Subconjunto de problemas en los que no existe un algoritmo que calcule el resultado exacto en tiempo polinomial.

Sin embargo, como se ha mencionado anteriormente, es probable que se desee hacer una reasignación de recursos parauna solución sub-óptima, i.e. un número de colores mayor al mínimo cromático). El problema resultante, llamado problema de coloración robusta, se ha utilizado para modelar la programación de tareas con probables cambios en la asignación de recursos.

Algunos ejemplos de aplicación de este tipo de planteamiento se relacionan con la asignación de microprocesadores en servidores web (donde se someten a abruptos cambios por la demanda de cada aplicación y de los usuarios), la asignación de salones y profesores en diferentes cursos, e incluso con la planificación de vuelos como se menciona en[6], siendo altamente requeridas soluciones eficientes.

2.3.Trabajos anteriores relacionados.

2.3.1. Literatura externa ala UAM.

Como antecedentes, se encuentra un amplio estudio de técnicas heurísticas aplicadas al PCR, y los resultados han sido publicados en trabajos externos como en [6], donde se propone un algoritmo memético, es decir un algoritmo evolutivo en el que el operador de mutación se sustituye por un operador de búsqueda local. En el trabajo mencionado, los autores proponen específicamente dos buscadores locales (búsqueda enumerativa y búsqueda estocástica), mismos que son comparados mediante experimentos numéricos en [7] y [8].

2.3.2. Proyectos relacionados en la UAM Azcapotzalco.

En tantoa la investigación generada en la casa de estudios UAM, el problema de coloración robusta es una variante del problema de coloración de grafos planteado por Javier Yáñez y el profesor de la UAM-Azcapotzalco Javier Ramírez [2]. En este trabajo, los autores proponen algunos algoritmos de resolución (enumeración parcial y un algoritmo genético combinado con un método glotón). El Dr. Pedro Lara Velázquez presentó de igual forma en su tesis doctoral dos algoritmos que resolvieran el problema de coloración robusta, particularmente GRASP y Búsqueda Dispersa[4].

Otros de los algoritmos empleados para la solución del problema de coloración robusta es el de recocido simulado presentado en [7]. Finalmente, en la UAM-Azcapotzalco se ha realizado dos trabajos de Proyecto Terminal, tratando el problema por medio de un método de búsqueda tabú combinado con un algoritmo glotón [5] y de un algoritmo exhaustivo para la coloración de árboles binarios [9].

3. Genetic Local Search (GLS).

El algoritmo GLS es un método heurístico híbrido, que combina las características de un algoritmo evolutivo y de una búsqueda local clásica. Esta técnica ha sido empleada anteriormente para problemas tales como el agente viajero, produciendo regularmente buenos resultados [10] y [11]. Cabe mencionar que el algoritmo de GLS aún no ha sido aplicado al problema de coloración robusta de gráficas.

3.1.Algoritmo Evolutivo.

El algoritmo evolutivo empleado en este trabajo se asemeja al implementado con anterioridad para el problema de coloración clásica, en el trabajo de Galinier y Hao [1].Se reutiliza el esquema de representación de los individuos (soluciones) basada en particiones. De acuerdo a este formalismo, una solución se representa mediante una lista de conjuntos de vértices, cada conjunto siendo asociado a una clase (color).

El operador de cruza está inspirado en la mitosis celular, partiendo de la idea de que se requieren dos individuos para producir uno o varios individuo(s) nuevo(s), que comparte(n) características de ambos padres. En este proyecto, el operador *Greedy Crossover*, propuesto en [1], fue adoptado. Sin embargo, este método es claramente enfocado al problema de coloración clásica ya que promueve una reducción sistemática del número de clores usados. Esta forma de operar no resultó tan eficiente para el problema de coloración robusta por lo que el operador fue entonces ligeramente modificado, con la ideade mantener constante el número de colores y así reducir la rigidez asociada (paso 4 del algoritmo descrito a continuación).

El algoritmo de cruza empleado en el proyecto es como sigue:

1. Se eligen dos individuos de manera aleatoria.
2. Mientras *i* sea menor que el número de clases:
    a. La clase más grande de uno del primer padre se hereda al hijo y se eliminan los nodos de esta clase de ambos padres.
    b. *i* se incrementa en uno.
3. Los nodos aún no asignados se agregan de manera aleatoria a alguna clase del hijo.
4. Si existen clases vacías en el hijo, se inicializa j=0 y se realiza lo siguiente para cada clase vacía:
    a. Se toma un vértice de la clase *j*en caso de que contenga más de un vértice.
    b. Se incrementa *j* en uno.

A su vez, el operador de mutación permite generar variaciones locales de losindividuos producidos, con el objetivo de evitarconvergencia prematura hacia óptimos locales y sin destruir soluciones buenas en cada iteración.La técnica de mutación empleada en este trabajo

consiste en tomar un nodo perteneciente a una clase e insertarlo en una clasedistinta.Las clases y nodos se seleccionande manera aleatoria.

Estos operadores en conjunto forman de manera general al algoritmo evolutivo. Cabe agregar que se usó una técnica de selección probabilística, con la clásica ruleta de Golberg. Finalmente, un factor muy importante para el desempeño del mismo el ajuste de los parámetros de funcionamiento:las tasas de supervivencia (determinando el número de individuos que sobreviven de una generación para otra y el número de hijos producidos en cada generación, de mutación (la probabilidad de que un individuo sufra una mutación), así como el número de individuos en cada población y el número de generaciones calculadas deben ser analizados en términos de eficacia y eficiencia.En secciones posteriores, se presentan con más detalle los estudios preliminares realizados para ajustar estos parámetros.

3.2.Buscador Local.

El buscador local (o *hill-climber*, por sus características glotonas) es una heurística empleada para resolver problemas como el presentado en este proyecto (NP-Duro).Muy propensa a quedar atrapada en óptimos locales, esta técnica ha sido mejorada de diferentes formas (recocido simulado y búsqueda tabú son los ejemplos más relevantes). Sin embargo, en general cualquier buscador local podría ser usado, ya que el propósito de introducir este operador al algoritmo evolutivo es mejorar los individuos de la población en cada generación, de tal forma que cada solución manejada por el algoritmo evolutivo sea un óptimo local. Debe considerarse también que la implementación de un algoritmo de búsqueda local más complejo incrementará el tiempo de cómputo total del GLS.

En el buscador local simple implementado en este trabajo, se genera, a partir de la solución actual *S*, una solución nueva en su entorno (i.e., una solución vecina), *S'*; se acepta *S'* sólo si mejora el valor de la función objetivo.El diseño del vecindario, o entorno, de una configuración*S* dada consiste en mover un solo nodo *v* de una clase a otra clase *V*.

El criterio de paro, expresado en términos del número de iteraciones consecutivas sin mejora del objetivo, es el único parámetro de operación a ajustar para este algoritmo.

3.3.Hibridación.

La mezcla o hibridación de los algoritmos expuestos es simple, pues el buscador local es considerado un operador más del algoritmo evolutivo.Por lo cual, a cada individuo de la población generado por mutación, se le aplicará el método de buscador local para mejorar la calidad de la población.

4. Desarrollo e implementación.

El presente trabajo se realizó mediante el paradigma de programación orientado a objetos; se plantearon clases que representaran los procesos del algoritmo así como los objetos que interactúan en ellos. Se desarrollaron todas las clases que se encuentran en el aplicativo del algoritmo en el lenguaje de programación Java.

A continuación se describen los algoritmos particulares del GLS, estos algoritmos dentro de la representación orientada a objetos fueron descritos en métodos, dentro de una clase global.

4.1.Implementación del modelado del PCR.

Es importante tener en cuenta que un individuo representa una solución al problema instanciado.Bajo esta representación, se define el planteamiento de una clase que tiene como datos miembro el conjunto de características propias de una solución, como son los nodos que este posee, ubicados en cada una de sus clases coloreadas, así también el número de conflictos que esta solución tiene entre sus vértices y la rigidez total. A su vez se considera una aptitud (fitness) que corresponde a la siguiente expresión:

$$\varphi = kC^2 + R$$

Donde,

$\boldsymbol{\varphi}$ indica la aptitud (fitness) del individuo,

$\boldsymbol{k}$ es la constante de penalización,

$\boldsymbol{C}$ es el número de conflictos que genera la solución entre sus vértices y,

$\boldsymbol{R}$ representa la rigidez total de la solución.

Para el presente proyecto se utilizó $k = 7$, dado que en las pruebas esta constante no marcaba grandes diferencias para números de mayor y menor magnitud.Simplemente se penalizan las soluciones que cuentan con conflictos, es decir infactibles, promoviendo una reducciónde l númerode conflictos hasta que éste se anule.Posteriormente en el proceso de búsqueda, cuando la población es completamente factible, la aptitud se basa en la pura rigidez.

4.2.Desarrollo e implementación del Algoritmo Evolutivo.

Como se mencionópreviamente, el algoritmo evolutivo implementa una técnica de cruza inspirada del algoritmo de Galinier y Hao [1], ligeramente modificada.Se profundizó particularmente en temas relacionados conel ajuste de la tasa de supervivencia (y de recombinación, que es su completo binario).

La elección de una tasa adecuada de supervivencia no puede ser ignorada para un desempeño satisfactorio del algoritmo. Se consideran los tipos de supervivencia mencionados en [12]: uno de ellos, denotado como de tipo *k*, sugiere que se adopte una supervivencia alta con una baja tasa reproductiva como es el caso de los mamíferos entre otras especies;con la estrategia de tipo *r*, al contrario,no hay un alto nivel de supervivencia sino un alto nivel reproductivo. Para este último caso considérese como ejemplo a los insectos.

Una vez identificadas estas estrategias de supervivencia, es importante aterrizar cuál de ellas se utilizará en el GLS, para ello es importante ver que a menor supervivencia la variación de las características se perderán a lo largo de su evolución y es posible perder buenas soluciones de manera repetida.Por ello se adoptó una estrategia similar a la de tipo *k*, dado que se cuenta con un operador de mutación aunque deja abierto a futuras investigaciones explorar ambas estrategias combinadas en el proceso de cruza.

Otro aspecto importante dentro de la implementación desarrollada es el hecho de que en el algoritmo de cruza se utilizó una selección aleatoria entre los padres (cruza sin elitismo), en futuras investigaciones es otro ámbito que puede profundizarse. Sin embargo, un método de selección más complejo dentro de la cruza implicaría más cómputo y esto podría convertir al GLS en un algoritmo más lento.

Si la selección de los padres es aleatoria, la selección de los supervivientes a nivel de población es probabilística, es decir que da a soluciones buenas una mayor probabilidad de sobrevivir. El método elegido se basa en la *ruleta de Goldberg*, de esta manera se obtienen las nuevas poblaciones en cada generación.

Finalmente, la tasa de mutación debe ajustarse de forma que permita visitar regiones inexploradas del espacio de búsqueda, sin tener un poder destructivo demasiado alto que impida o frene la convergencia.

A continuación se muestra el algoritmo evolutivo desarrollado en el presente proyecto:

1. Se genera aleatoriamenteuna población inicial con individuos, acordes al problema instanciado.
2. Desde 0 hasta el Número máximo de generaciones – 1 se realiza lo siguiente:
    a. Se obtiene el mejor individuo y se almacena en una lista.
    b. Desde 0 hasta el número máximo de apareamientos permitidos –1:
        i. Se seleccionan dos padres de manera aleatoria y se cruzan; el individuo obtenido (hijo) se almacena en una lista de individuos.

c. Desde 0 hasta el número máximo de supervivientes–1:
    i. Se selecciona un individuo mediante la *ruleta de Goldberg* y se almacena en la lista donde se almacenaron los hijos.
d. Se encima la lista nueva en la lista vieja de población y se elijen de manera aleatoria un número de individuos que son modificados mediante el proceso de *mutación*.

Como se puede observar, el desempeño del Algoritmo Evolutivo depende directamente del tamaño de población y el número de generaciones. Para lo que fue necesario hacer un conjunto de pruebas para poder adoptar los parámetros que mejor se ajustaban haciendo un equilibrio entre la calidad de las soluciones y el tiempo de cómputo.

4.3.Desarrollo e implementación del Buscador Local.

Como se expuso anteriormente, el buscador local utilizado es un método de tipo*hill-climber*, que en este proyecto se implementó de la siguiente manera:

1. Se toma un individuo.
2. Mientras N sea mayor que 0
    a. Se toma un nodo de una de sus clases de manera aleatoria y se evalúa la solución
    b. Si la solución es mejor entonces sustituye a la original y N se reinicializa a 0.
    c. De lo contrario, entonces N se incrementa en 1 y el individuo no cambia.

En un principio se consideróconstruir cada vecino realizando varios cambios consecutivos, sin embargo pruebas preliminares demostraron que esto no era conveniente ya que las soluciones no mejoraban. Entonces, un vecino está construido en base a un único movimiento (como para la mutación).

4.4.Implementación del GLS.

El funcionamiento del algoritmo híbrido se representa en la figura 1 de la página siguiente. Como bien se puede observar, el Algoritmo Evolutivo es la base del GLS, pues la hibridación consiste simplemente en incluir el buscador local como un operador genético adicional.

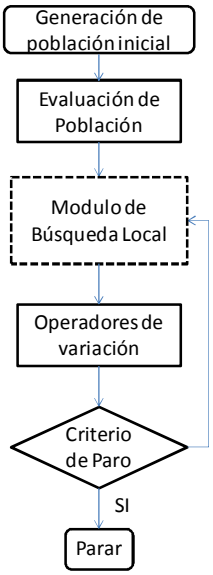


Figura 1. Diagrama de flujo del GLS

El pseudo-código asociado se proporciona a continuación:

1. Se genera una población inicial con individuos con características aleatorias acordes al problema instanciado.
2. Desde 0 hasta el Número máximo de poblaciones – 1 se realiza lo siguiente:

    a. Se obtiene el mejor individuo y se almacena en una lista.
    b. Se aplica el método de búsqueda local a cada individuo.
    c. Desde 0 hasta el número máximo de apareamientos permitidos –1:
        i. Se seleccionan dos padres de manera aleatoria y se cruzan; el individuo obtenido (hijo) se almacena en una lista de individuos.
    d. Desde 0 hasta el número máximo de supervivientes - 1:
        i. Se selecciona un individuo mediante la *ruleta de Goldberg* y se almacena en la lista donde se almacenaron los hijos.
    e. Se encima la lista nueva en la lista vieja de población y se elijen de manera aleatoria un número de individuos que son modificados mediante el proceso de *mutación*.

Y así es como se obtiene el algoritmo *Genetic Local Search* aplicado al problema de coloración robusta.

5. Experimentación.

Los experimentos numéricos efectuados se basaron en un banco de problemas propuesto en [10], cuyas características se describen en la siguiente tabla:

Problema	# vértices	# colores
Al20	20	8
Al30	30	10
Al40	40	14
Al50	50	17
Al60	60	20
Al70	70	24
Al80	80	27
Al90	90	30
Al100	100	34
Al110	110	37

Tabla 1. Instancias resueltas en este proyecto

5.1.Ajustes de parámetros.

Anteriormente se mencionaron algunos parámetros a ajustar, que incluso dieron lugar a modificaciones en los procesos internos del algoritmo. Dentro de la experimentación descrita en esta sección, se puede observar la sensibilidad del algoritmo de GLS a estos parámetros.

Los primeros parámetros estudiados fueron el tamaño de la población y el número de generaciones. Se notó que el tiempo de cómputo se impacta de manera más significativa al incrementar el número de individuos en la población, así que fue crucial reducir este parámetro. Pruebas preliminares indicaron que el mejor valor era de 80 individuos.

Para la determinación del número de generaciones, era importante determinar la eficiencia del algoritmo con respecto a diferentes tamaños de instancias. Se determinó que este valor debería de variar según la instancia.Para el problema de al20 se calcularon 150 generaciones mientras que para el problema de al110, este valor subió a 400 generaciones, debido a la complejidad creciente de las instancias.

En el siguiente diagrama, se presentan las pruebas hechas para cada instancia según el número de generaciones necesarias (en promedio) para encontrar la mejor solución:

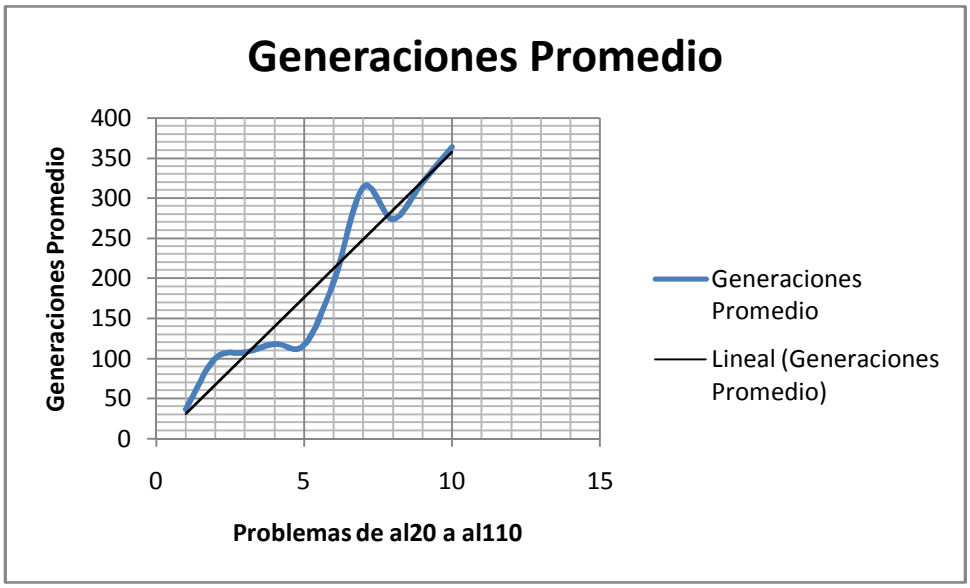


Figura 2. Generaciones promedio para cada una de las instancias.

De igual manera, se obtuvieron valores relevantes para los demás parámetros para el algoritmo, para concluir en el siguiente ajuste para el Algoritmo Evolutivo:

Tamaño de la población, 80 Individuos.
Número de generaciones: dependiendo del tamaño de la instancia tratada, entre 150 y 400 generaciones.
Tasa de mutación de 0.05.
Constante de penalización, 7.0.
Tasa de recombinación: 0.25
Tasa de Supervivencia: 0.75 (= 1 – Tasa de recombinación).
Número de iteraciones para el buscador local: 50

En cuanto al buscador local, el número de iteraciones sin mejora autorizadas, es decir el número de vecinos generados hasta concluir que la solución actual es localmente óptima, se efectuaron también pruebas preliminares (por separado de las del Algoritmo Evolutivo), concluyéndose que el criterio de paro debería ser igual a 400 ya que por su comportamiento asintótico en la mejora de soluciones, más allá de este número de generaciones no existen cambios en la mejora de los individuos.

5.2.Resultados.

Los resultados para cada una de las instancias fueron los indicados en latabla 2. Esta tabla presenta, para cada instancia resuelta y cada algoritmo, el valor de la mejor solución obtenida y el tiempo de cómputo asociado (en segundos).

Problema	GLS	GRASP	Búsq. dispersa
Al20	4.6934 /6.766	4.6934 / 0.15	4.6934 / 1.3
Al30	7.5749 / 11.5	7.5749 / 0.44	7.5749 / 3.6
Al40	7.41 / 15.5	7.3950 / 1.0	7.0837 / 8.5
Al50	9.9654 / 20.5	8.9531 / 1.9	8.2587 / 13
Al60	12.9101 / 26	9.9687 / 3.3	8.8676 / 19
Al70	12.6806 / 51.2	11.2388 / 5.31	9.2634 / 36
Al80	15.2734 / 83.4	11.7512 / 8.0	9.9835 / 52
Al90	17.7636 / 98	13.4919 / 11.5	10.8911 / 47
Al100	17.3336 / 112	12.8675 / 15.8	10.8911 / 47
Al110	20.4192 / 142.5	12.7681 / 20.7	10.8463 / 149

Tabla 2. Resultados del GLS con el banco de instancias propuesto

Estos resultados contemplan la mejor solución obtenida con 10 corridas del algoritmo de GLS (con semillas diferentes), esto debido que el algoritmo es estocástico.El tiempo expuesto en los resultados es un promedio obtenido de las corridas mencionadas. Nótese igualmente que debe ser considerado que el lenguaje de programación usado fueJava, lo que facilita la modelación de problemas pero con un sacrificio en tiempo de cómputo.

Se nota que el algoritmo de GLS proporciona soluciones de buena calidad para instancias pequeñas (gráficas de hasta 40 vértices). Sin embargo, esta calidad se va deteriorando al tratar instancias de tamaño medio y grande. La mejor solución obtenida en estos casos resulta mucho más alta que los resultados obtenidos por la técnica de Búsqueda Dispersa, para tiempos de ejecución similares.

6. Conclusiones.

El algoritmo implementado, en comparación con otroscomo Búsqueda Dispersa o GRASP, no resultó ser tan eficaz en determinar solucionesde buena calidad. Sin embargo, este trabajo deja pendientes muchos aspectos que pueden ser profundizados en futuras investigaciones para su mejora. Así como la mejora tecnológica en el modelado y la programación que para poder dar eficiencia al tiempo, se podría realizar un modelado que contemple el uso de hilos para que algoritmo sea ejecutado en menor tiempo, con la ayuda de las actuales generaciones de procesadores con varios núcleos.

Se observó una diferencia significativa entre las pruebas realizadas con los algoritmos de buscador local y evolutivo por separado y las pruebas con el algoritmo de GLS (híbrido entre ambos algoritmos). La estrategia de combinar técnicas heurísticas para poder obtener mejores soluciones mostró ser muy alentador. El esquema de hibridación requiere sin embargo de un estudio meticuloso basado en cada una de las técnicas, ya que a veces el uso de técnicas más complicadas puede causar aumentos en los tiempos de cómputopor poca mejora en la calidad de las soluciones.

Entre otras cosas, el proyecto deja en evidencia que la importancia de evaluar el problema y así saber qué tan crítico es el tiempo de cómputo contra la calidad de las soluciones que pueda devolver la heurística.Algoritmos no deterministas, como el presentado en este trabajo de proyecto terminal, pueden ser eficientes en uno de los sentidos al que uno quiera adaptarlos.

El problema de coloración robusta, como otros problemas, deja abierto estas disyuntivas. El uso adecuado de heurísticas para resolverlos y aplicarlos a problemas cotidianos en los cuales es esencial tener soluciones que de buena calidad y en poco tiempo.

Bibliografía

[1] P. Galinier y J. Hao, Hybrid evolutionary algorithms for Graph Coloring, 1999.

[2] J. Yañez y J. Ramírez, The robust coloring problem, 2003.

[3] J. Ramírez, Extensiones del Problema de Coloración de Grafos, Tesis de Doctorado., Noviembre 2000.

[4] Pedro Lara Velázquez, "Un algoritmo evolutivo para resolver el problema de coloración robusta", Revista de Matemática, Vol 12, Nos. 1 y 2, pp. 111-120, Junio 2005.

[5] I. Frausto Benítez, Implementación de un algoritmo de búsqueda tabú para resolverel problema de coloración robusta, Agosto 2009.

[6] F. Wang, A. Lim and S. Gou Y. Kong, A new Hybrid Genetic Algorithm for the Robust Graph Coloring Problem, 2003.

[7] P. Lara Velázquez, S.G. de los Cobos Silva M. A. Gutiérrez Andrade, A new Simulated Annealing algorithm for the Robust Coloring Problem, July 2007.

[8] Pedro Lara Velázquez, Rafael López Bracho y Javier Ramírez Rodríguez Miguel Ángel Gutiérrez Andrade, Heuristics for the robust coloring problem, 2010.

[9] C. A. Rodríguez Villalobos, Algoritmos para coloración robusta de árboles binarios, Agosto 2008.

[10] P. Merz y B. Freisleben., Genetic Local Search for the TSP: New Results, Abril 1997.

[11] E.H.L. Aarts, H.J. Bandelt, P.J.M. van Laarhoven y E. Pesch N.L.J. Ulder, “Genetic Local Search algorithms for the traveling salesman problema.

[12] Robert H. MacArthur, *Geographical Ecology: Patterns in the Distribution of Species*. USA: Princeton University Press, 1972.

[13] R. M. Brady, Optimization Strategies Gleaned from Biological Evolution, 1985.

[14] M. Gorges-Schleuter, ASPARAGOS: An Asynchronous Parallel Genetic, 1989.