UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

SECRETARÍA DE ESTUDIOS DE POSGRADO

FACULTAD DE INGENIERIA

UN ALGORITMO EVOLUTIVO PARA RESOLVER EL PROBLEMA DE COLORACIÓN ROBUSTA

T E S I S

COMO REQUISITO PARA OBTENER EL GRADO DE

DOCTOR EN INGENIERIA

(INVESTIGACION DE OPERACIONES)

Presenta

PEDRO LARA VELÁZQUEZ

DIRECTOR DE TESIS:

DR. MIGUEL ANGEL GUTIERREZ ANDRADE

CIUDAD UNIVERSITARIA
JUNIO, 2005

VNIVERSIDAD NACIONAL AVFNOMA DE MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA

VOTOS APROBATORIOS

DR. WILFRIDO RIVERA GÓMEZ FRANCO
Coordinador del Programa de Posgrado
en Ingeniería, U N A M
P r e s e n t e

Por este medio comunico a usted que he leído la tesis titulada **"UN ALGORITMO EVOLUTIVO PARA RESOLVER EL PROBLEMA DE COLORACION ROBUSTA"** para obtener el grado de DOCTOR EN INGENIERIA en el campo del conocimiento **INGENIERIA SISTEMAS**, que presenta el alumno: **PEDRO LARA VELAZQUEZ.**

Al mismo tiempo me permito informarle mi decisión de otorgar o no el voto aprobatorio.

JURADO		VOTO APROBATORIO		FIRMA	FECHA
PRESIDENTE	DR. SERGIO FUENTES MAYA	(SI)	(NO)		6-05-05
VOCAL	DR. MIGUEL ANGEL GUTIERREZ ANDRADE	(SI)	(NO)		16-05-05
VOCAL	DR. SERGIO DE LOS COBOS SILVA	(SI)	(NO)		16-05-05
VOCAL	DR. JAVIER RAMIREZ RODRIGUEZ	(SI)	(NO)		11-5-05
SECRETARIO	DRA. IDALIA FLORES DE LA MOTA	(SI)	(NO)		6-05-05
SUPLENTE	DR. RICARDO ACEVES GARCIA	(SI)	(NO)		6-V-05
SUPLENTE	DR. JOSE DE JESUS ACOSTA FLORES	(SI)	(NO)		6-V-05

BJS*jac.

Agradecimientos de la Tesis

La persona que agradece de corazón algún bien que le hacen, se sentirá feliz y será acreedora al cariño de quienes se lo hacen y aún de las que le conocen.

Mª Guadalupe Valero. *Diario Escolar.*
San Luis Potosí, Miércoles 17 de enero de 1913

Al **Dr. Miguel Ángel Gutiérrez Andrade**, por sugerir este tema de tesis y por los dos maravillosos cursos que he tomado con usted. Este trabajo doctoral no hubiera sido posible sin su apoyo, revisión, consejos y regaños, todos en el momento oportuno. Creo que no hay palabras para agradecerle todo lo que ha hecho por mí, pero, una vez mas, le doy mil gracias por todo, de todo corazón.

Al **Dr. Javier Ramírez Rodríguez**. Por todas sus sugerencias y comentarios sobre este documento, por facilitarme ejecutables de sus algoritmos para Coloración Robusta con Búsqueda Tabú, Enumeración Parcial, Cotas y por facilitarme varios artículos y referencias relacionados a la investigación.

Al **Dr. Sergio Fuentes Maya.** Por sus sugerencias en lo que a la estructura del documento de Tesis concierne. Por revelarme un punto de vista diferente de la vida, que de otra forma no hubiera conocido. Con el debido respeto, por todo lo que compartimos y no compartimos.

Al **Dr. Sergio de los Cobos Silva**. Por sus comentarios en el contenido del documento de tesis, por sus consejos sobre la preparación y desarrollo de los artículos y por aceptar ser sinodal.

Al **Dr. José de Jesús Acosta Flores**, **Dra. Idalia Flores de la Mota** y al **Dr. Ricardo Aceves García,** por aceptar ser sinodales de este trabajo.

A la **M. en A. P. Ma. del Pilar Alonso Reyes** y al **M. en C. José Antonio Flores Díaz**, por permitirme tomar los cursos de Probabilidad I, Probabilidad II, Estadística I y Análisis de Regresión, calificar mis tareas y permitirme tener los mismos derechos y obligaciones que los alumnos regulares en sus cursos. Son y serán un ejemplo de lo que los grandes profesores pueden y deben ser. De todo corazón, fue un privilegio tomar clase con ustedes.

A **Fred Glover PhD, Manuel Laguna PhD** y al **Dr. Rafael Martí Cunquero,** por crear el sendero por el que he llevado mi vida durante los últimos dos años: *"Caminante, son tus huellas el camino y nada más... caminante, no hay camino, se hace camino al andar..."*

To **J. E. Gutierrez Chavez MIB, Cristina Serverius MA, Siti-Hajar Salleh MSc** and **Beatriz E. Fuentes Madariaga PhD,** for your time and good advice when reviewing my work in English. Few people like you in this world!

A la **DGEP** por otorgarme una beca durante tres (de los cuatro y medio) años del doctorado. Especialmente a la **M. en E. Artemisa Pedroza de Gortari** y a la **Srita. Alva Linda Cruz**, por hacerme la burocracia más sencilla. También a la **Ing. Ma. Antonieta Teyssier Borja** por simplificar los tramites de apoyo económico para asistir al INFORMS 2004.

This dissertation is dedicated:

I'm looking through you... Where did you go?
I thought I knew you... What did I know?
You don't look different but you have changed...
I'm looking through you... You're not the same!

***James Paul McCartney**, October of 1965*

Cleaned a lot of plates in Memphis
Pumped a lot of pain down in New Orleans
But I never saw the good side of the city
Until I hitched a ride on a river boat queen
Big wheels keep on turning
Proud Mary keep on burning
Rolling, rolling, rolling on the river

***John Fogerty**, autumn of 1968*

To my beloved wife, **Gabriela Torres Servin**, for your love and friendship, and for your support throughout my doctoral dissertation. I met you on a November evening... after fourteen years and a zillion adventures, we are still arguing for the big and small things... thank you...

To my parents, **Pedro Lara Reyes** and **Martha Teresa Velázquez Chavira.** I have always thought that a parent's biggest legacy is a remarkable example to follow, an excellent genetic mix, and in some few occasions, the solidarity to rescue each other from an out-of-control adventure. As I grown old, I find that the freedom (beyond all doubt) that you gave me to be myself, and your unconditional love and tenderness, makes me who I am... Thank you guys! You're the best parents ever!

To **Miguel Angel Gutierrez Andrade**, my "academic dad", a great human being and an example of what amazing people can do with life. I consider myself very fortunate for having taken classes and for having worked with you. I still got a lot to learn from you. Thank you so much Doctor!

To **Beatriz Elizabeth Fuentes Madariaga**, my "academic mom", for your occasional advice, for letting me be myself in the Lab most of the time, and for your continuous support.

To my good old four female cats: **MaChi, MaChancha, MaNegra and Chanchota**, for their biggest virtues: *MaChi*, the hunter, *MaChancha*, the mother, *MaNegra*, the enigma, and *Chanchota*, the wise woman.

To **Juan Enrique Gutiérrez Chávez**, for St Peter's Cathedral (I'd rather have Amsterdam's Town Hall, but... that's life...). An incarnation of my mom wandering through Europe, following his own destiny... that's really cool!!!

To **Juan Enrique Gutiérrez Velázquez**, for those creepy travels to Morelia and Zacatecas.

To **Siti-Hajar Salleh,** a brave young woman from the other side of the world. You kept encouraging me to finish my work, and gave me hope. You are right behind me, struggling down the same road. Less than a year to go gal... almost there! All my good vibes to you... keep up the good work! Terima kasih, Hajar!!!

To **Chimaru-San, Indy-San, Rule-San, Cameha-San, Lord Marcus, Chucho, Humberto, Astrid and Lala**. Thanks for sharing your wonderful and instructive "collective trip" with me; always interesting, but never EVER edificatory... Especially to **Edgar Chimal,** for the *Wikipedia*, the *Mozilla Firefox* and the *ZipGenius*. Also to **Fernando Del Río,** for some priceless media files, fundamental in my metamorphosis from the analog incarnation to the digital format. Indy-San is the living proof that droids can be more humane than some so-called "humans".

Do I contradict myself?
Very well, then, I contradict myself.
I am large, I contain multitudes.

Walt Whitman, *Song of Myself, 1885*

This is also dedicated to **George Martin, Paul McCartney, John, George & Ringo, Robert Allen Zimmerman, Scott Joplin, J.S. Bach, J.C.W.A. Mozart, Charles Spencer Chaplin, Jean-Michel Jarre, Benito Pablo Juarez García, Leonardo Gomes-Navas Chapa, Concepción Ruiz Ruiz-Funes, Dmitrii Kouznetzov Kallistratova, Ruben Tellez Sanchez, Magdalena Carmen Frieda Kahlo y Calderón, Julio Cortazar, David Byrne, Peter Garrett, Thomas Jeffrey Hanks, Arthur Charles Clarke, Richard Phillips Feynman, Alexander Sutherland Neil, Carl Edward Sagan**, and the 400,000 people team in the **Apollo Project** effort, for showing me that the key to building my own destiny, is faith, some creativity and lots and lots of perseverance...

C'mon People Let The Fun Begin.
We've Got A Future And It's Rushing In.
Call All The Minstrels From The Ancient Shrine.
Pass Down The Message, That It's Right This Time...

Índice

Abstract

The Robust Coloring Problem (RCP) is a generalization of the Graph Coloring Problem, in which we seek for a sturdy coloring that remains valid when extra edges were added. The RCP is used to schedule events susceptible to last minute changes, stable frequency assign of the electromagnetic spectrum and robust register allocation in microprocessors. It has been proved that it is a NP-Hard kind of Problem and heuristic techniques are needed for big instances. In this research two algorithms were developed: a GRASP and a Scatter-Search/GRASP Algorithm. Both of them were tested against some proposed instances used in previously developed algorithms such as Simulated Annealing, Taboo Search, Genetic and Partial Enumeration Techniques. Our GRASP is the fastest, with high quality solutions and our SS/GRASP Algorithm gives the best known solutions to the problem.

Keywords: Graph Coloring, Robust Coloring, Scatter Search, Heuristics.

Resumen

El Problema de Coloración Robusta (PCR) es una generalización del problema de coloración de grafos, donde el objetivo es crear una solución que permanezca válida al agregar aristas adicionales. El PCR se utiliza para calendarizar eventos con posibles cambios de ultimo momento, asignación estable de frecuencias del espectro electromagnético y para la asignación robusta de registros en microprocesadores. Este problema es del tipo NP-Duro y para instancias grandes es necesario utilizar técnicas heurísticas. En este trabajo se desarrollaron dos algoritmos: un GRASP y un Algoritmo de Búsqueda Dispersa. Se comparan los resultados experimentales obtenidos con ambos algoritmos propuestos, contra técnicas creadas con anterioridad, tales como Recocido Simulado, Búsqueda Tabú, Algoritmos Genéticos, y Enumeración Parcial. Nuestro GRASP es el más rápido de los algoritmos que generan soluciones de alta calidad y nuestro Algoritmo de Búsqueda Dispersa da las mejores soluciones al día de hoy para el problema.

Palabras Clave: Coloración de Grafos, Coloración Robusta, Búsqueda Dispersa, Heurísticas.

Introducción.

Ground rules.- The greatest objection that has been raised about our lunar rendezvous plan is that it does not conform to the "ground rules". This to me is nonsense; the important question is, "Do we want to get to the moon or not?", and, if so, why do we have to restrict our thinking along a certain narrow channel. I feel very fortunate that I do not have to confine my thinking to arbitrarily set up ground rules which only serve to constrain and preclude possible equally good or perhaps better approaches.

***John C. Houbolt**, Associate Chief.*
NASA Langley Research Center. November 15, 1961

La optimización está íntimamente ligada a la raza humana desde sus orígenes: siempre es deseable realizar una actividad consumiendo el mínimo de recursos o maximizando el beneficio por su realización. Ya sea una pequeña tribu utilizando al máximo sus reservas de comida para sobrevivir el invierno hace 20,000 años, o medio millón de personas administrando sus recursos para llegar a la Luna en un tiempo mínimo, es evidente que la optimización es parte de la naturaleza humana.

Los problemas de optimización matemática se dividen en dos categorías: problemas con variables continuas y problemas con variables discretas. Estos últimos son llamados problemas de optimización combinatoria. Por ejemplo, deseamos visitar n ciudades, y queremos encontrar en que orden lo debemos hacer para que la distancia recorrida o el costo final sean mínimos. Resolviendo un modelo conocido como el Problema del Agente Viajero podemos encontrar dicho recorrido. Si lo que deseamos es minimizar el número de metros de cable requeridos para abastecer de electricidad a un conjunto de poblaciones, lo podemos modelar como un Árbol de Expansión Mínima. Estos son solo un par de ejemplos de los muchos que existen dentro del mundo de la Optimización Combinatoria.

Los problemas combinatorios se pueden dividir en "fáciles" y "difíciles": los fáciles son aquellos para los que se conoce un algoritmo que se ejecuta en "tiempo polinomial"[1] para encontrar el valor óptimo; ejemplos de estos problemas combinatorios son el Árbol de Expansión Mínima y el de Ruta Más Corta. Los problemas difíciles son aquellos en los cuales no se ha encontrado un algoritmo de solución acotado por una función polinomial, y se les ha llamado NP-Duros. En estos problemas a medida que el tamaño de la instancia aumenta, el tiempo de ejecución para encontrar el valor óptimo se vuelve prohibitivamente grande; ejemplos de este tipo son el Problema del Agente Viajero y el Problema de la Mochila.

[1] Es decir, existe un algoritmo de solución que obtiene el valor óptimo y su tiempo de ejecución respecto del tamaño de la instancia está acotado por una función polinomial.

Como ejemplo de estos problemas "NP-Duros", en [Gutiérrez, 1991] se menciona que si tenemos un problema con *n!* soluciones (*n* es el tamaño de la instancia) y si pudiéramos explorar un billón de ellas por segundo, para *n=23* la computadora terminaría su tarea en 820 años y para *n=24* terminaría en 19,624 años. Los algoritmos exactos para problemas NP-Duros no se encuentran muy lejos de estos números. Es más práctico tener una técnica que permita encontrar un conjunto de muy buenas soluciones y no estar en la búsqueda de "El Óptimo", el cual puede dejar de serlo al introducir algún cambio en nuestro modelo matemático.

Un problema combinatorio NP-Duro muy famoso es el Problema Generalizado de Coloración, en donde dado un cierto grafo, si dos vértices comparten una arista, estos vértices deben estar pintados con distintos colores. El Problema de Coloración Mínima (PCM) consiste en encontrar una coloración válida utilizando el mínimo de colores. Este enfoque no toma en cuenta modificaciones, ya que al agregar alguna arista adicional al modelo, la coloración se puede invalidar, tirando por la borda todo el trabajo previo.

El Problema de Coloración Robusta (PCR) toma en cuenta esta última posibilidad, ya que además de encontrar una coloración válida (con el mínimo de colores o no), se busca que dicha coloración sea robusta, es decir, que sea una coloración "dura de matar" al agregar aristas adicionales. Algunas de las aplicaciones más usuales son la calendarización de eventos, asignación de frecuencias y asignación de memoria en microprocesadores. Se ha demostrado que el PCR es también un problema NP-Duro [Ramírez, 2003], por lo que el uso de un método aproximado, es necesario a medida que aumenta el número de vértices. En este sentido, para encontrar buenas soluciones, aunque no necesariamente óptimas de los problemas NP-Duros, se han desarrollado las técnicas llamadas heurísticas[2] y meta-heurísticas.

Las Técnicas Heurísticas nos permiten encontrar una buena solución, usando una regla de sentido común. Por ejemplo, si quiero comprar el "mejor traje de vestir", un criterio heurístico de compra sería adquirir el traje mas caro, o el de la marca más prestigiada; el traje adquirido puede o no ser "el mejor traje" (Aunque habría que especificar en qué consiste ser "el mejor"), pero muy probablemente adquiriríamos un traje de tela agradable al tacto, hipoalergénica y de buen aspecto, esto sin hacer un estudio profundo sobre calidad de telas, las alergias comunes en el país o la moda de vestir actual.

Las heurísticas son tan antiguas como el Universo mismo[3], pero la invención de las técnicas heurísticas se atribuye al matemático George Polya, el cual en su libro de 1945 *"How to Solve It"*, menciona: "El objetivo de las heurísticas es estudiar los métodos y reglas de los descubrimientos y las invenciones", además dice que las heurísticas deben ser "rápidas y

[2] Heurístico viene de la misma raíz griega que la palabra "eureka" ("ευρισκω"), que significa "encontrar", "heurístico" podría traducirse como "lo que se utiliza para descubrir".

[3] Los átomos al combinarse, buscan encontrarse en estados de "casi" mínima energía (valores muy cercanos al valor de mínima energía). Por otra parte, la vida en el planeta Tierra ha llevado a cabo un experimento en Optimización Combinatoria Heurística sobre el ADN que ha durado más de 4 mil millones de años y ha logrado la biodiversidad actual del planeta Tierra.

frugales". Un ejemplo de las heurísticas aplicadas a problemas combinatorios son los algoritmos glotones, donde se van acumulando soluciones sin recapacitar en las decisiones previas que se han tomado.

Las meta-heurísticas son técnicas que, como su nombre lo indica, van más allá de los resultados "rápidos y frugales" de las heurísticas. Son verdaderos "planes de batalla" con los cuales, tomamos los resultados de una heurística y mejoramos estas soluciones. Ejemplos de estas técnicas son la Búsqueda Local, GRASP, Algoritmos Genéticos, Recocido Simulado, Búsqueda Tabú y Búsqueda Dispersa.

Una técnica que recientemente ha gozado de gran popularidad, son los algoritmos GRASP[4], que obtienen buenas soluciones con rapidez. Se usan para obtener una solución rápida, o bien, para generar soluciones iniciales de alguna meta-heurística más compleja. Esta técnica trabaja en dos fases: una fase de construcción y una de mejora. Sus resultados son buenos y rápidos, pero técnicas mas complejas obtienen mejores resultados. Una de las mejores y más recientes de las metaheurísticas "complejas" es la técnica de Búsqueda Dispersa.

La Técnica de Búsqueda Dispersa fue propuesta en su forma actual en [Glover, 1998]. Esta técnica ha sido utilizada en una gran variedad de problemas, principalmente en modelos combinatorios y no lineales[5]. La Búsqueda Dispersa es una técnica evolutiva que construye nuevas soluciones combinando los mejores resultados previos.

Regresando al tema de esta investigación, el problema de Coloración Robusta se ha estudiado para instancias pequeñas, tanto con técnicas exactas como con técnicas heurísticas, hasta un tamaño máximo de 30 vértices. Es deseable desarrollar un par de algoritmos, uno que encuentre buenas soluciones muy rápidamente y otro que sea altamente eficiente, y que encuentre las mejores soluciones posibles en instancias grandes.

Un GRASP se podría ocupar de instancias de hasta mil vértices, gracias a su rapidez. Por otro lado, un Algoritmo de Búsqueda Dispersa nos podría ofrecer las mejores soluciones en instancias de cien y quizás hasta ciento cincuenta vértices. En ambos algoritmos el trabajo se justifica por los buenos resultados que han tenido estas técnicas en problemas combinatorios en general, y para el problema de coloración mínima en particular como se ve en [Laguna, 2001] y [Hamiez, 2002].

[4] GRASP viene de "Greedy Randomized Adaptive Search Procedure" y fue desarrollada por Feo y Resende en 1995. Para una referencia completa de sus aplicaciones se puede consultar [Resende, 2003].

[5] Para una referencia completa de los problemas resueltos mediante esta técnica se pueden consultar las referencias [Glover, 1999] y [Glover[a], 2004].

Presentación de la Investigación

El objetivo general de este trabajo fue desarrollar dos algoritmos eficientes que resuelven el Problema de Coloración Robusta de Grafos. Los objetivos particulares son:

- Generar un Algoritmo Glotón Mejorado (GRASP) que encuentre soluciones rápidamente para instancias muy grandes.
- Desarrollar un Algoritmo de Búsqueda Dispersa para instancias grandes.
- Extender el estudio del Problema de Coloración Robusta para instancias desde 30 hasta más de 100 vértices con Búsqueda Dispersa y para instancias hasta 1000 vértices con el GRASP.
- Implementar en la computadora códigos eficientes para ambas técnicas que resuelvan el Problema de Coloración Robusta.
- Motivar y difundir las técnicas GRASP y Búsqueda Dispersa para su aplicación en problemas combinatorios en general y en el Problema de Coloración Robusta en particular.
- Motivar y difundir el Problema de Coloración Robusta como una alternativa más apegada a la realidad que el conocido Problema de Coloración Mínima.

Este trabajo está dividido en seis capítulos:

El Capítulo 1 trata sobre el Problema de Coloración Robusta. Se menciona primero el problema de coloración mínima, que es el fundamento original del PCR. Después se hace una revisión del problema de Coloración Robusta y se dan dos ejemplos de aplicación.

El Capítulo 2 muestra los modelos y técnicas de solución utilizadas previamente para resolver el PCR. Se explica un Modelo Exacto Binario, y los algoritmos heurísticos de Enumeración Parcial, Genético, Búsqueda Tabú y Recocido Simulado, además, se presenta un algoritmo que puede generar cotas superior e inferior del valor mínimo de rigidez de la coloración de un cierto grafo. Finalmente se presenta un sumario de los resultados obtenidos con ellos.

El Capítulo 3 presenta el marco teórico de las técnicas GRASP y Búsqueda Dispersa aplicadas a problemas combinatorios. Se explican las dos fases de la técnica GRASP y los cinco elementos que componen la Búsqueda Dispersa, así como sus respectivos algoritmos genéricos. Se muestran finalmente ejemplos orientados a problemas combinatorios.

En el Capítulo 4 se explica el diseño de las técnicas GRASP y de Búsqueda Dispersa para resolver el Problema de Coloración Robusta, objetivo central de este trabajo. Se hace además un estudio sobre cuáles fueron los procedimientos y parámetros más eficientes que se encontraron.

En el Capítulo 5 se muestra la implementación de los algoritmos. Se presentan diagramas y pseudo códigos de ambos, se muestra la convergencia de soluciones en el Algoritmo de Búsqueda Dispersa y un ejemplo de aplicación paso a paso en una instancia robusta pequeña que se resuelve con ambas técnicas.

En el Capítulo 6 se presenta la experiencia computacional obtenida con los algoritmos realizados en esta investigación y se hace un estudio comparativo de los mismos contra la experiencia previa en Coloración Robusta y en Coloración Mínima, observándose que el algoritmo de Búsqueda Dispersa realizado, da las mejores soluciones conocidas al día de hoy para el Problema de Coloración Robusta, que ambos tienen un buen desempeño para encontrar coloraciones mínimas; además que el GRASP genera soluciones de alta calidad mas rápido que cualquier otro algoritmo anterior.

Al final, se dan las conclusiones y se muestran las aportaciones más importantes de este trabajo.

Capítulo 1. Problema de Coloración Robusta.

El Problema de Coloración Robusta es utilizado para la asignación estable de frecuencias (en teléfonos celulares, internet inalámbrico, de frecuencias en control de tráfico aéreo, etc.); para problemas de calendarización que permiten cambios posteriores (asignar el número de bloques de tiempo en que se utilizarán máquinas o recursos, o el número de días en que se realizarán las conferencias en un congreso, etc.); y para asignación robusta de registros (cómo utilizar la memoria interna de un microprocesador minimizando el uso o sin recurrir a la memoria externa RAM y así realizar más rápidamente los cálculos).

El Problema de Coloración de Grafos asocia a un conjunto de eventos, llamados vértices (frecuencias, asignaturas, recursos, etc.) un grupo de incompatibilidades, llamadas aristas (cuando dos asignaturas que no se pueden impartir simultáneamente, dos recursos que no se pueden utilizar simultáneamente, etc.), con vértices y aristas generamos un grafo. Si dos vértices comparten una arista, dichos vértices deben de ser pintados con colores diferentes. El objetivo es pintar todos los vértices del grafo sin incompatibilidades de colores.

En el Problema de Coloración Robusta, además de pintar válidamente los vértices, se busca estabilidad, es decir, la coloración se debe mantener vigente ante la inclusión de aristas adicionales en el grafo. Este enfoque es más apegado a la realidad porque es común que una vez que se ha encontrado una solución, se hacen modificaciones, y cualquier arista adicional al grafo lo puede invalidar. Al buscar la coloración más robusta, buscamos la coloración menos susceptible a modificaciones para una cantidad de colores dada, sea el mínimo de colores o no. Dicho de una manera más formal, el Problema de Coloración Robusta busca pintar válidamente un grafo, minimizando la suma de los pesos de las aristas del grafo complementario cuyos vértices comparten el mismo color.

En este capítulo se hace una revisión del problema de Coloración Robusta de Grafos. En la sección 1.1 se presenta el problema de Coloración Mínima, el cual es el antecesor directo del Problema de Coloración Robusta; en la sección 1.2 se hace una revisión del problema de Coloración Robusta, con un ejemplo ilustrativo. En la Sección 1.3 se presentan dos ejemplos de aplicación. Definiciones de los términos utilizados en este capítulo se encuentran en el apéndice.

1.1. Antecedentes: Problema de Coloración Mínima.

El Problema de Coloración Mínima (PCM) es uno de los problemas más conocidos en la Optimización Combinatoria. En este problema, dado un cierto grafo, si dos vértices comparten una arista, estos vértices deben estar pintados con colores diferentes. El PCM consiste en encontrar una coloración válida utilizando el mínimo de colores.

Ejemplo de Calendarización de Exámenes.

Supongamos que estamos haciendo los horarios de exámenes finales del semestre y queremos evitar que un alumno haga más de un examen cada día. Designemos a los siete cursos por *A, B, C, ... G.* En la *Tabla 1.1* un asterisco en la entrada *i, j* significa que el curso *i* y el curso *j* tienen, por lo menos, un estudiante en común, y por tanto, no se puede calendarizar el mismo día. ¿Cuál es el número mínimo de días necesarios para programar todos los exámenes?

Tabla 1.1. Si dos cursos comparten al menos un estudiante, se prohíbe realizar el mismo día el examen (se prohíbe que ambos vértices tengan el mismo color)

	A	B	C	D	E	F	G
A		*	*	*		*	*
B	*		*				*
C	*	*		*			
D	*		*		*	*	
E				*		*	
F	*			*	*		*
G	*	*				*	

Veamos cómo modelar este problema como uno de coloración mínima: tenemos los cursos *A, B, ... G* y una tabla en la cual si hay un asterisco, entonces dos estudiantes comparten el mismo curso y por lo tanto no pueden hacer el examen el mismo día. Haremos un grafo donde los vértices representan los cursos, y si dos cursos comparten por lo menos un estudiante, dichos cursos se conectan con una arista. Si coloreamos ese grafo, dos cursos que son adyacentes tendrán distintos colores. El objetivo es utilizar el mínimo de ellos.

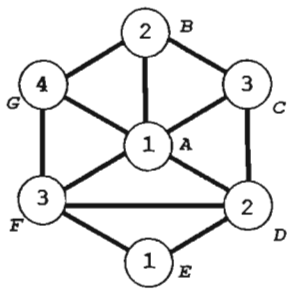


Figura 1.1. Grafo asociado al ejemplo del Problema de Calendarización: cada vértice simboliza un curso, y cada arista representa al menos un estudiante que presenta dos exámenes.

Como el grafo es pequeño, podemos resolverlo por ensayo y error, y encontraremos que su número cromático es 4, es decir, necesitamos por lo menos cuatro días para efectuar los exámenes.

Tabla 1.2. Calendarización con el mínimo de días, asociada al grafo de la figura 1.1.

Día 1	Examen A y E
Día 2	Examen B y D
Día 3	Examen C y F
Día 4	Examen G

Si generamos un vector donde la primera entrada representa el día en que se realizará el examen *A*, la segunda cuando se realizará el *B* etc, el vector de coloración (cada color indica en que día se aplicara el examen) es:

(1,2,3,2,1,3,4)

Esta es una solución posible, pero existen otras más. Dicho sea de paso: Aquí estamos considerando el mínimo de colores (días), pero si nos dan toda la semana (5 días = 5 colores), tendríamos muchas más opciones de solución, y podríamos considerar algún otro factor para realizar nuestra programación de exámenes.

En este ejemplo es fácil encontrar una solución por ensayo y error, pero para instancias más grandes se requieren algoritmos especializados. Para medir la habilidad de los algoritmos para encontrar coloraciones mínimas y con que rapidez, se han desarrollado instancias de prueba como las que se describen a continuación.

Instancias Clásicas de Coloración Mínima. (Grafos DIMACS)

Para probar le eficiencia de los algoritmos que resuelven el problema de coloración Mínima, se desarrollaron en 1993 las primeras instancias DIMACS[1]. Desde entonces hasta ahora se han agregado nuevas instancias y en la actualidad de todas ellas se sabe su número cromático o al menos una cota superior del mismo. Estas instancias son muy útiles ya que están basadas en instancias reales o bien, son grafos diseñados específicamente para ser difíciles de resolver.

En prácticamente todos los artículos recientes en donde se trata heurísticamente este problema se incluyen estos grafos (por ejemplo, [Ribeiro, 1999], [Galinier, 1999], [Eiben, 1998], [Laguna, 2001] y [Hamiez, 2001] por citar algunos). Estas instancias se pueden obtener de la página de Michael Trick: *"Graph Coloring Instances"*:

[1]Las iniciales DIMACS vienen de "Center for Discrete Mathematics and Theoretical Computer Science", y fueron propuestos originalmente por David Johnson (AT&T Bell Laboratories) y Michael Trick (Universidad Carnegie Mellon) para motivar el uso del Problema de Coloración Mínima.

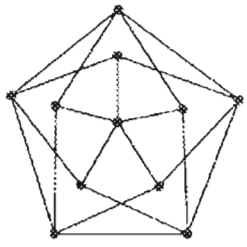
http://mat.gsia.cmu.edu/COLOR/instances.html

Se describen brevemente los grafos utilizados en este trabajo:

- MYC: Estos grafos son difíciles de colorear mínimamente ya que están libres de triángulos (no tienen clanes de orden 3), y el número cromático se incrementa al aumentar el tamaño del problema. En la *figura 1.2a* se muestra un grafo de este tipo. (instancias *Mycielski 3-7*)

- SGB: En esta base de datos encontramos cuatro tipos de grafos:
  - *Grafos de libros.* Dado un trabajo de literatura, creamos un grafo donde cada vértice es una letra o símbolo. Dos vértices están conectados por una arista si los caracteres correspondientes se encuentran uno al lado del otro en el texto. Se usan los grafos de cuatro libros clásicos: *Anna Karenina* de Leon Tolstoi; *David Copperfield* de Charles Dickens; *Huckleberry Finn* de Mark Twain; y *Les Miserables,* de Víctor Hugo.

  - *Grafos de Reinas.* Dado un tablero de ajedrez de tamaño $n \times n$, un grafo de Reinas es un grafo donde cada vértice corresponde a un cuadrado del tablero. Dos vértices están conectados por una arista, si los cuadrados correspondientes están en la misma fila, columna o diagonal. Este problema tiene una interpretación natural: dado un tablero, ¿es posible colocar n reinas de tal forma que no se encuentren en la misma fila, columna o diagonal? La respuesta a esta pregunta es afirmativa, si se puede pintar válidamente con n colores ($n \geq k$). en la *figura 1.2b* se muestra la solución para el caso de un tablero de 5x5, (instancias *Queen Graph 5x5-14x14*)



(a)
Ejemplo de grafo libre de triángulos

5	4	3	2	1
3	2	1	5	4
1	5	4	3	2
4	3	2	1	5
2	1	5	4	3

(b)
Una coloración válida del grafo de reinas, caso n=5

Figura 1.2. Ejemplo de un grafo tipo Mycielski y solución asociada a colorear un tablero de reinas con n=5

  - *Grafos de Distancias.* Los vértices representan un conjunto de ciudades de Estados Unidos y dos vértices están unidos por una arista si la distancia entre ambas ciudades es menor o igual que una cierta cota. Aunque este ejemplo no tiene un aplicación directa, está basado en datos reales de distancias y se pueden generar conglomerados con distintos umbrales de distancia α. Se dan

las siguientes cotas: α = 250, 500, 750, 1000 y 1500 millas. (instancias *Miles Graph 250-1500*)

- *Grafos de Juegos.* En este grafo, cada vértice representa un equipo en la Liga de Fútbol Americano Colegial de EUA. Si dos equipos compitieron entre sí durante la temporada, son unidos por una arista. Este grafo se basa en datos reales de la liga de 1990. (instancias *College Football*).

SCH: Grafos de calendarización de cursos en la Universidad de Wisconsin, cada curso es un vértice y cada arista un estudiante. Se consideran dos casos: incluyendo salas de estudio y sin ellas (instancias *Class Scheduling*).

REG: Problemas basados en asignación de registros para variables en programas reales (instancias *zeroin.i.1-3, mulsol.i.1-5, fpsol2.i.1-3, inithx.i.1-3*).

LEI: Grafos Leighton con número cromático garantizado (instancias *Leighton 450 5-25*).

1.2. Problema de Coloración Robusta: Definición y Ejemplos.

Este problema, que es un caso particular del Problema Generalizado de Coloración de Grafos, se justifica en [Ramírez, 2000] de la siguiente forma:

> *"En algunas circunstancias, el planteamiento del mínimo número de colores no es crítico, sino que interesa una solución al problema que sea estable, en el sentido que al añadir o cambiar aristas al grafo, la coloración continúe siendo válida. Estas consideraciones muestran que el problema de coloración mínima es muy restrictivo para este tipo de problemas".*

En el modelo de Coloración Robusta (PCR), consideramos dos grafos, el original, el cual se debe pintar válidamente, y su grafo complementario asociado (es decir, todas las aristas no incluidas en el grafo original). Cada arista en el grafo complementario tiene una penalización (por ejemplo, la posibilidad de que esa arista posteriormente sea incluida en el grafo original), si ambos extremos de la arista complementaria comparten vértices con el mismo color, la penalización correspondiente a esa arista se agrega a la función de rigidez.

Para las definiciones formales de los términos asociados, se puede consultar el apéndice al final de este documento.

Así, el PCR considera a la rigidez del grafo complementario como una especie de función objetivo por minimizar, y la validez de la coloración en el grafo original cumple las veces de un conjunto de restricciones.

Tomemos por ejemplo, el problema de asignación de registros de memoria, en donde el número de colores utilizado está asociado al número de registros que se necesitan simultáneamente para ejecutar algún programa. Supongamos que tenemos una cámara digital cuyos programas de codificación y decodificación requieren únicamente 69 y 97

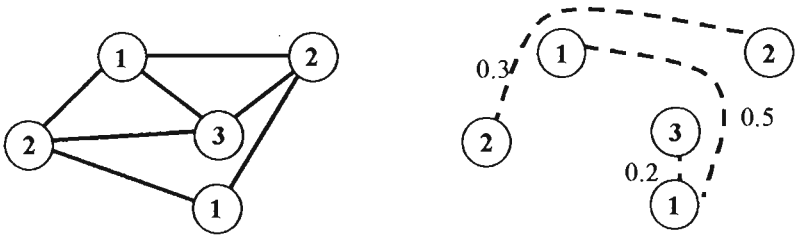
"colores" cada uno. Como el microprocesador solo se vende con un tamaño fijo de 128 registros[2], buscando minimizar el número de registros de memoria estamos dejando sin utilizar el 46% y el 24% de los mismos respectivamente. ¿Por qué no en estos casos, se utilizan 128 "colores" para pintar el grafo asociado y se busca una coloración que minimice los cambios entre los modos de codificación y decodificación? Así, de todas las coloraciones posibles del grafo asociado, algunas generan menos cambios que otras. La que genera menor modificaciones es la coloración más robusta.

El PCR se puede reducir al Problema de Coloración Mínima, cuando pintamos el grafo con el número cromático de colores (la mínima cantidad de colores que permite una coloración válida en el grafo original), y todas las penalizaciones en el grafo complementario las hacemos cero. El PCM es un caso especial del PCR.

Un ejemplo ilustrativo para el Problema de Coloración Robusta.

Supongamos que hay cinco aeronaves en una cierta región geográfica[3]. Si dos aviones están los suficientemente cerca, y sus equipos estuvieran sintonizados a la misma frecuencia, se causarán interferencia entre ellos, por lo que se deben utilizar distintas frecuencias.

En un modelo de coloración de grafos asociado a este problema, si dos aeronaves se causan interferencia, debemos unir ambos vértices (aviones) con una arista y, en ese caso, debemos usar distintos colores (distintas frecuencias) entre ellas. Considérese el grafo de la *figura 1.3a,* como el modelo de interferencia actual entre los equipos y su grafo complementario penalizado (*figura 1.3b*), nos muestra la probabilidad de que en los próximos veinte minutos otros aviones se interfieran entre sí.



(a) *Grafo Original*

(b) *Grafo complementario penalizado*

Figura 1.3. Coloración válida en el grafo original y una rigidez en el grafo complementario de 0.8, utilizando 3 colores.

[2] Debido a que el número de registros con que se venden los procesadores es en tamaños 2^n, es decir, 2, 4, 8, 16, 32, 64, 128, etc.

[3] El ejemplo es igualmente valido si consideramos usuarios de teléfonos celulares, radios de dos vías o internet inalámbrico, o cualquier otro grupo de usuarios del espectro electromagnético.

Podemos pintar este grafo con un mínimo de tres colores (debemos utilizar 3 frecuencias como mínimo) como se muestra en la *figura 1.3* (los números indican un color). Para esta coloración, las aristas del grafo complementario penalizadas con 0.3 y 0.5 contribuyen a la rigidez (ambos extremos de esta arista comparten el mismo color); así, en esta configuración, tenemos una rigidez de 0.8. Una solución alternativa, mostrada en la *figura 1.4,* que utiliza los mismos tres colores y una rigidez de tan solo 0.5, se muestra a continuación:

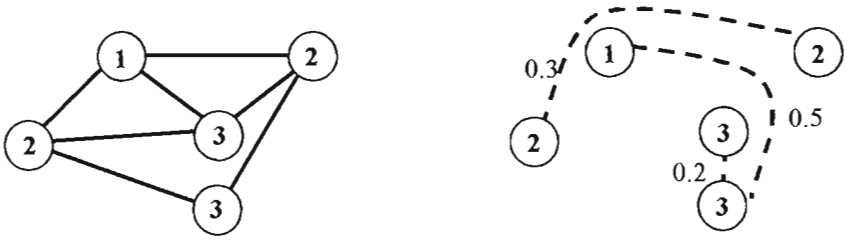


Figura 1.4. Coloración válida en el grafo original y una rigidez de 0.5, utilizando tres colores

Finalmente, si se nos permite utilizar cuatro frecuencias, la rigidez disminuye hasta 0.2:

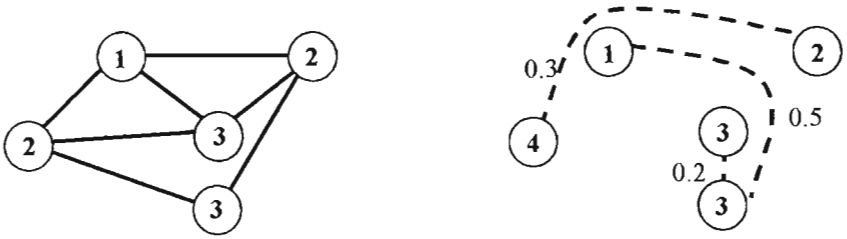


Figura 1.5. Coloración válida en el grafo original y una rigidez de 0.2, que utiliza cuatro colores

Obviamente, si en este grafo de cinco vértices se nos permite utilizar cinco colores, la rigidez se vuelve cero.

1.3. Aplicaciones. ([Ramírez, 2000] y [Ramírez 2003])

Como se ha dicho anteriormente, el problema de coloración robusta implica dos aspectos: una coloración obligatoria que debe ser satisfecha (no necesariamente mínima) y una coloración "deseable" que incluya la minimización (maximización) de rigidez en el grafo complementario, según sea el caso. Los siguientes ejemplos de aplicación siguen esta idea.

Programación de Exámenes.

De nuevo en [Ramírez, 2000] se define este problema:

> *"Se necesita programar los exámenes en una Universidad. Si dos asignaturas comparten al menos un alumno, no se programarán el mismo día. Teniendo en*

cuenta esta restricción, se puede construir el grafo de incompatibilidad G = (V, E) cuyos vértices son las asignaturas. La arista {i, j} que pertenece a E indica que hay al menos un estudiante inscrito en las asignaturas i y j que pertenecen a V.

Éste es un problema típico de planificación de recursos, los elementos a planificar son las asignaturas y el recurso que determina la incompatibilidad entre las asignaturas es el día en que se realizarán los correspondientes exámenes".

Supongamos que asociado a nuestro problema debemos programar 6 exámenes diferentes (*a, b, c, d, e* y *f*), con el grafo de incompatibilidades mostrado en la *figura 1.6a* y su grafo complementario asociado mostrado en la *figura 1.6b*, aun sin considerar penalizaciones.

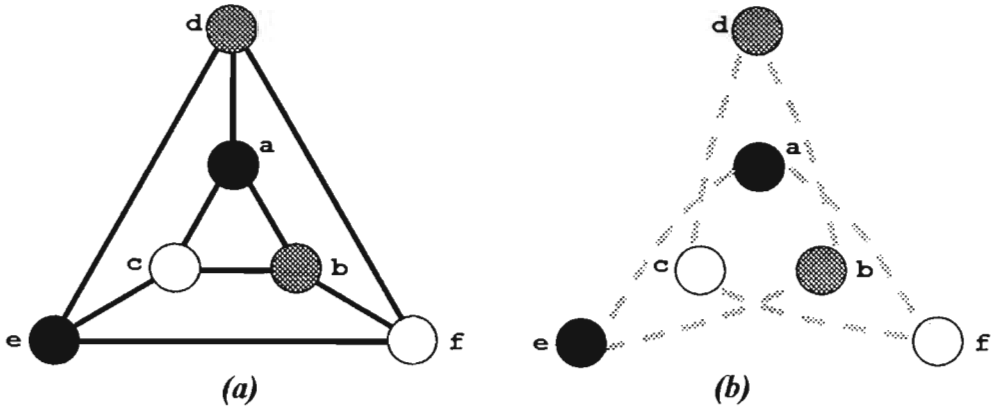


Figura 1.6. Grafos original y complementario (sin penalizaciones todavía), para un problema de programación de exámenes.

Para este grafo, el número cromático es 3 (es decir, tres colores es el número mínimo de colores para pintarlo). En nuestro problema esto quiere decir que se requieren por lo menos tres días para realizar los exámenes.

Ahora, supongamos que una vez fijado el calendario un alumno cambia una asignatura no prevista por la coloración de nuestro modelo (por ejemplo, que ahora sus materias sean la *b* y la *d*), nuestra coloración se invalidaría. Mediante una coloración robusta buscaríamos minimizar la posibilidad de que esto ocurra.

Supongamos que hay 50 alumnos, cada uno de los cuales escoge sólo dos materias y que el número de alumnos en cada asignatura es el siguiente:

$$n_a = 5\ ;\ n_b = 30\ ;\ n_c = 10\ ;\ n_d = 30,\ n_e = 20\ ;\ n_f = 5$$

si consideramos que la posibilidad de que ocurra un cambio aumenta con el número de alumnos inscritos n_k (es decir, es más probable que en un grupo grande alguno deserte, o que llegue un nuevo alumno desde otro grupo), es un buen criterio considerar que las probabilidades p_{ij}, de añadir la arista *{i, j}* del grafo complementario aumentan con n_i y n_j. Podemos construir la siguiente tabla donde tenemos además de los productos $n_i \times n_j$, incluimos las posibilidades normalizadas:

Tabla 1.3.

$\{i,j\} \in \overline{E}$	$n_i \times n_j$	P(i,j)
{a,e}	100	0.0506
{a,f}	25	0.0127
{b,d}	900	0.4557
{b,e}	600	0.3038
{c,d}	300	0.1519
{c,f}	50	0.0253

Y teniendo 3 días para programar los exámenes, el problema se podría enunciar de la siguiente manera:

> *"Encontrar una 3-coloración que minimice la probabilidad de añadir una arista con los dos extremos con la misma coloración."*

Lo que buscamos es una coloración que siga siendo válida al añadir alguna arista. Suponiendo la independencia de estos sucesos, es decir, el cambio de asignatura, la probabilidad (es decir, minimizar la rigidez, ver apéndice) de que se añada una arista al grafo complementario que no invalide la coloración, es:

$$\prod_{(i,j)\in\overline{E},\ C(i)=C(j)} (1-p(r_{ij}))$$

En nuestro ejemplo, con las penalizaciones consideradas tenemos:

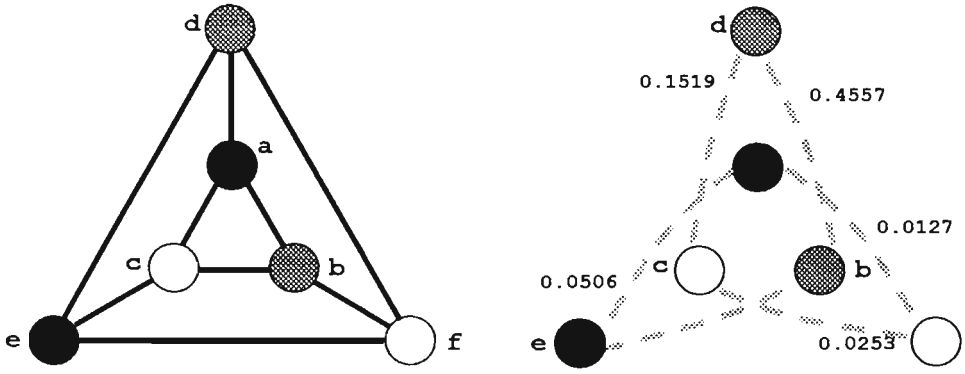


Figura 1.7. Grafo pintado con tres colores, con las penalizaciones asociadas al ejemplo de programación de exámenes, en las aristas del grafo complementario, con función objetivo de 0.5037

Con esta coloración obtenemos:

$$(1\text{-}0.0506)*(1\text{-}0.4557)*(1\text{-}0.0253) = 0.5037$$

Ahora consideremos la coloración:

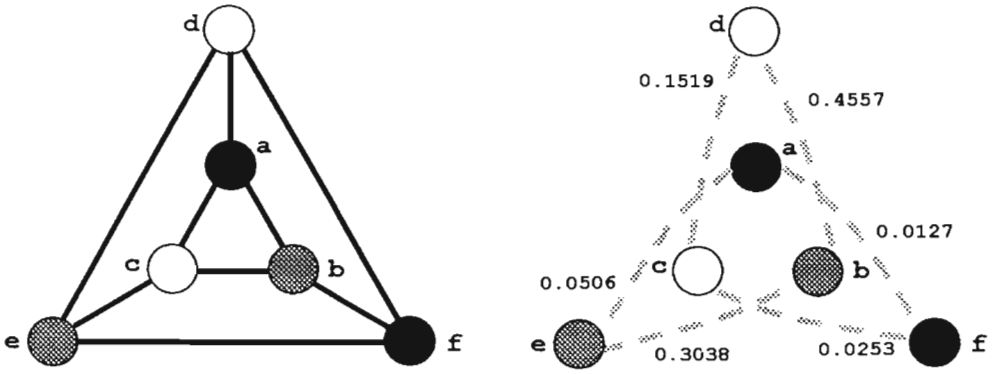


Figura 1.8. Grafo pintado con tres colores y probabilidad de 0.5829

La probabilidad es ahora:

$$(1-0.0127)*(1-0.3038)*(1-0.1519) = 0.5829$$

La cual es mejor a la anterior.

Si nos permiten programar los exámenes en cuatro días, podemos realizar la siguiente coloración:

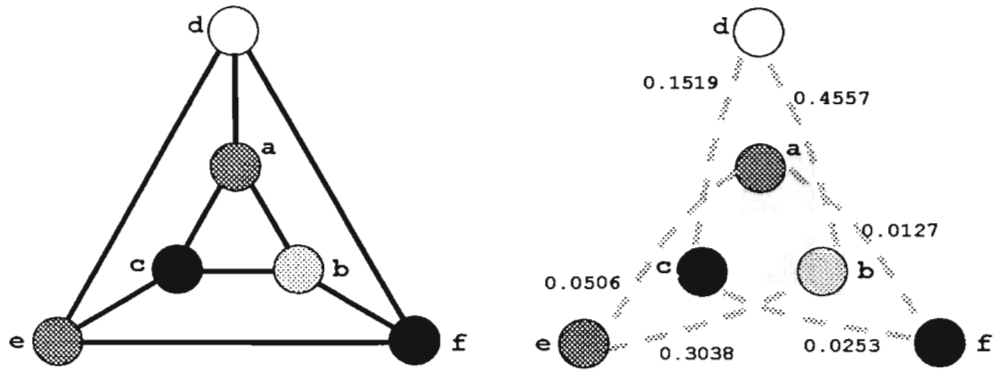


Figura 1.9. Grafo pintado con 4 colores, y una probabilidad de 0.9254

La probabilidad de que se conserve válida esta coloración es ahora:

$$(1-0.0506)*(1-0.0253)=0.9254$$

Como se ve en este problema de maximización de posibilidades, encontrar una coloración de mayor posibilidad de éxito puede aumentar al incrementar el número de colores disponibles.

Asignación de registros en microprocesadores
(Determinación de conglomerados, modificado de [Ramírez, 2000])

En el problema de conglomerados, particionamos un conjunto de elementos *{1,2,....n}* en clases, de forma que dos elementos de la misma clase están más cerca uno del otro, que dos elementos de clase diferente. Para determinar la "cercanía" se define una distancia *d* entre todos los elementos del conjunto y de un umbral $\alpha > 0$ fijado previamente: así, dos elementos se consideran próximos, si su disimilitud es menor o igual que dicho umbral α.

Podemos modelar este problema como uno de coloración, uniendo a dos elementos cuando su disimilitud sea mayor que el umbral α fijado ($d(i,j) > \alpha$). Cualquier coloración válida del grafo induce un conglomerado en el que las clases están definidas por los vértices que están igualmente coloreados. El menor número de clases del conglomerado se asocia así, al número cromático del grafo. Podemos buscar una coloración válida robusta que siga siendo válida para valores mayores del umbral α.

Ejemplo: Dentro de un microprocesador contamos con algunas celdas de memoria, llamadas registros, donde se guardan los valores de las variables de los programas. Si estos registros no son suficientes, se debe recurrir a la memoria externa, conocida comúnmente como RAM. El problema es que el tiempo requerido para asignar una variable a la memoria RAM es entre 10 y 100 veces mayor, por lo que es deseable tener el mayor número de registros, pero el microprocesador se vuelve mucho más caro a medida que aumentamos su número de registros.

Podemos generar una matriz de penalizaciones en donde encontramos el tiempo adicional de ejecución de un programa si las variables *i* y *j* comparten el mismo registro y dado el caso en que se usen simultáneamente, se deba "exportar" uno de ellos a la memoria RAM. El objetivo será entonces, encontrar una coloración de *n* variables en un programa, con *k* registros (colores) disponibles dentro del microprocesador y la rigidez asociada indica el tiempo adicional de ejecución al considerar esa coloración.

Consideremos una subrutina, la cual cuenta con 5 variables, con una matriz de tiempo adicional de ejecución (en segundos) como se muestra en la *tabla 1.4.*; supongamos que solamente tenemos dos registros para utilizarse al mismo tiempo:

Tabla 1.4. Matriz de tiempo adicional de ejecución

	a	*b*	*c*	*d*	*e*	
	–	0.01	0.02	0.05	0.04	*a*
		–	0.04	0.03	0.04	*b*
			–	0.06	0.07	*c*
				–	0.03	*d*
					–	*e*

Así, para distintos valores de α se obtienen distintos grafos y coloraciones. Por ejemplo, si se nos prohíbe tener tiempos de retardo superiores o iguales a $\alpha = 0.07$ *segs*, el grafo asociado a nuestro problema tiene una sola arista: *{c,e}*, como se muestra en la *figura 1.10*:

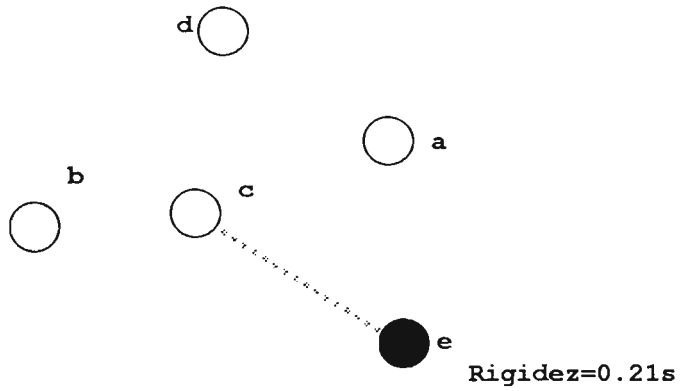


Figura 1.10. Coloración no robusta de dos registros que no tiene retrasos mayores $\alpha \geq 0.07$ segs. (arista punteada) con una penalización en tiempo (rigidez) de 0.21 segundos

Si consideramos el orden de los vértices *(a,b,c,d,e)* y haciendo *blanco=1* y *negro=2*, el vector de coloración asociado a la *figura 1.10* es *(1,1,1,1,2)*. Esta forma de asignar registros nos da una rigidez de 0.21. La rigidez en este problema representa el tiempo de ejecución adicional, en segundos, por cada ciclo del proceso.

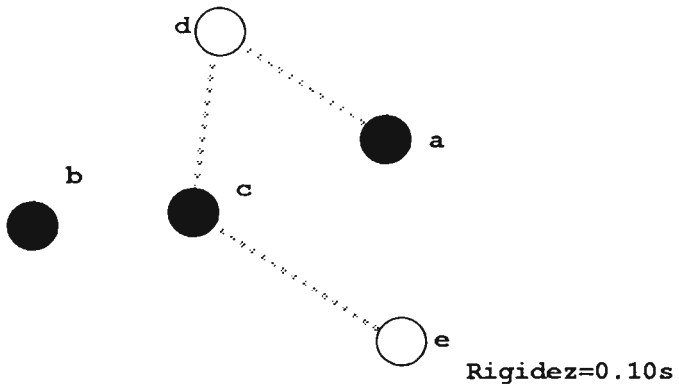


Figura 1.11. Coloración Robusta de también dos colores que no solo es válida para $\alpha = 0.07$ sino es lo suficientemente robusta para aceptar $\alpha = 0.05$ segs (aristas punteadas).

Considerando la coloración *(1,1,1,2,2) (figura 1.11)*, que utiliza los mismos dos registros, tenemos un retraso de poco menos de la mitad al reducirlo a 0.10 segundos de penalización por cada vez que se ejecuta la subrutina.

Si nos podemos dar el lujo de comprar un microprocesador con tres registros[4] (colorear el grafo con tres colores), una coloración robusta seria *(2,1,2,1,3)*, la cual tiene una rigidez de 0.05 segundos, como se muestra en la *figura 1.12*:

[4] Aunque la regla de 2^n registros es siempre valida de 8 registros en adelante, se suelen hacer microprocesadores con 3, 5 o 6 registros. En algunas otras ocasiones se rompe esta regla (por ejemplo, hay procesadores especializados en video con 144 registros).

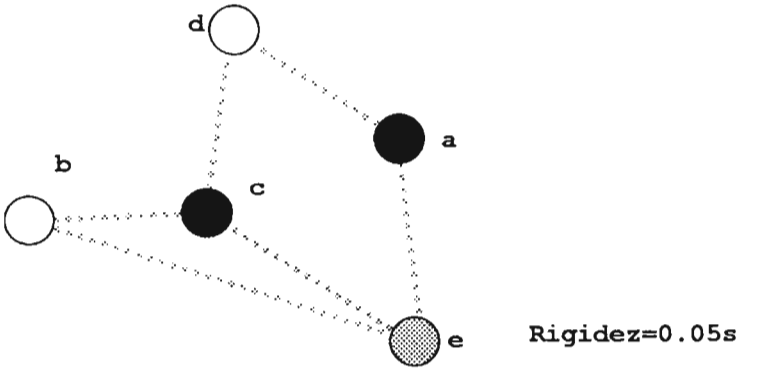


Figura 1.12. Coloración Robusta con tres colores, que no tiene retrasos mayores a 0.03 segundos por cada par de variables.

Esta coloración es tan robusta que sigue siendo válida para umbrales de $\alpha = 0.03$ *segs* (es decir, dadas dos variables, jamás se genera con ellas un retraso mayor a 0.03 segundos de ejecución de la subrutina). Las aristas punteadas son aquellas que tienen un retraso mayor a lo mencionado, y generan un "grafo original" válido.

Si ahora se nos permite pintar con 4 colores, podemos encontrar una coloración robusta con rigidez = 0.01 segundos, asociada a la *figura 1.13* cuyo vector de coloración sería *(1,1,2,3,4)*.

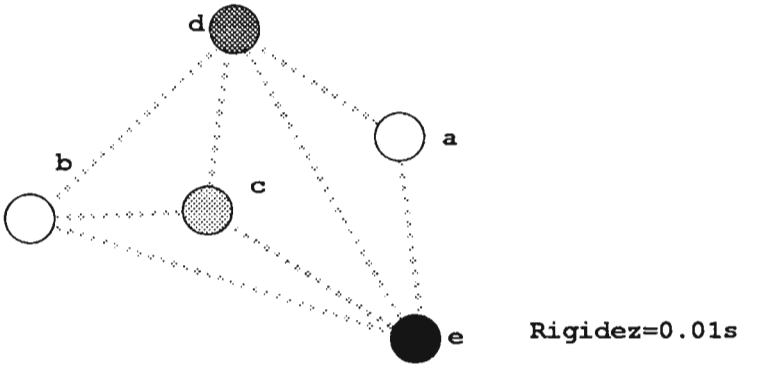


Figura 1.13. Coloración Robusta con cuatro colores, que jamás tiene penalizaciones entre variables mayores a 0.02 segs y tiene una rigidez de una centésima de segundo.

Si podemos costear microprocesadores que tienen 5 registros, nuestra subrutina no tiene tiempos de retraso, tenemos tantas variables como registros y, nuestro modelo en este caso, carece de sentido.

Una vez presentado el Problema de Coloración Robusta con un par de ejemplos, objetivo de este capítulo, el siguiente paso es mostrar que técnicas de solución que se han utilizado en el problema, el cual es el objetivo del próximo capítulo de esta tesis.

Capítulo 2. Métodos de Solución para el Problema de Coloración Robusta.

Para resolver el Problema de Coloración Robusta, se han propuesto diversos modelos, tanto exactos como heurísticos. Inicialmente se propuso un Modelo Exacto Binario, el cual requiere varias horas para encontrar el valor óptimo en instancias con 15 o 20 vértices. Para instancias más grandes es necesario el uso de técnicas heurísticas.

Las primeras técnicas heurísticas utilizadas fueron un Algoritmo Genético y una técnica Glotona de Enumeración Parcial. Aunque estas técnicas no encuentran soluciones tan buenas como el modelo exacto, son muy rápidas, y pueden obtener soluciones factibles, aunque poco robustas, para instancias de tamaños muy grandes, hasta unos 1,000 vértices.

Posteriormente se implementaron para el problema las técnicas de Búsqueda Tabú y Recocido Simulado, que en la mayoría de las instancias lograban encontrar la rigidez mínima obtenida con el método exacto. De todas las técnicas probadas previas a este trabajo de investigación, la de Recocido Simulado es la que encontraba los mejores resultados. En tiempos muy recientes, en [Ramírez, 2005] se trabaja en un modelo que genera cotas superior e inferior de rigidez para el valor óptimo, este modelo funciona muy bien para instancias con menos de 50 vértices.

En este Capítulo se hace una revisión general de los Modelos Matemáticos y las Técnicas de Solución Heurística, utilizadas con anterioridad a esta tesis para resolver el Problema de Coloración Robusta, y la experiencia computacional que existe. En la sección 2.1. se presenta un Modelo Lineal Binario, cuatro técnicas heurísticas de solución: Genéticos, Enumeración Parcial, Búsqueda Tabú y Recocido Simulado y un algoritmo que genera cotas superior e inferior de la rigidez mínima para una instancia dada. En la sección 2.2. se muestra un sumario del desempeño comparado del modelo, algoritmos y cotas, con respecto a ciertas instancias diseñadas específicamente para este problema.

2.1. Modelos y Técnicas de Solución. (Estado del Arte del PCR)

Los métodos siguientes suponen un grafo $G=(V,E)$ con n vértices y con un número de colores válido $k > 0$ fijo (k es mayor o igual al número cromático) y definidas las penalizaciones de las aristas del grafo complementario $\{p_{ij} \forall \{i,j\} \in \overline{E}\}$. Definiciones relacionadas a los términos usados en este capítulo se pueden encontrar en el apéndice.

Modelo de Programación Lineal Binaria

Descripción del Modelo. Este modelo fue propuesto originalmente en [Ramírez, 2000] y fue el primer método de solución utilizado para el problema. Es un modelo exacto de solución, y mediante su uso podemos obtener el valor mínimo de rigidez. Para definirlo, consideremos las siguientes variables de decisión:

$$x_{is} = \begin{Bmatrix} 1 \ si \ C(i) = s \\ 0 \ si \ C(i) \neq s \end{Bmatrix} \forall i \in \{1,2,...,n\}; \ \forall s \in \{1,...,k\}$$

Esta variable binaria nos determina el color de un cierto vértice: si $x_{is} = 1$, el vértice i está pintado con el color s. Si además introducimos la siguiente variable auxiliar:

$$y_{ij} = \begin{Bmatrix} 1 \ si \ existe \ s \in \{1,...,k\} \ tal \ que \ x_{is} = x_{js} \\ 0 \ en \ caso \ contrario \end{Bmatrix} \forall \{i,j\} \in \overline{E}$$

Con estas variables podemos plantear el PCR como uno de programación lineal binaria:

$$Min \sum_{\{i,j\} \in \overline{E}} p_{ij} y_{ij}$$

Sujeto a:

$\sum_{s=1}^{k} x_{is} = 1$	$\forall i \in \{1,...,n\}$	(Restricciones tipo 1)
$x_{is} + x_{js} \leq 1$	$\forall \{i,j\} \in E \ y \ \forall s \in \{1,2,...,k\}$	(Restricciones tipo 2)
$x_{is} + x_{js} - 1 \leq y_{ij}$	$\forall \{i,j\} \in \overline{E} \ y \ \forall s \in \{1,2,...,k\}$	(Restricciones tipo 3)

En las restricciones de tipo 1, nos aseguramos de asignarle a cada vértice únicamente un color. En las tipo 2 aseguramos que la coloración es válida; en la tercera aseguramos que si dos vértices $i, j \in V$ no unidos por una arista tienen el mismo color, entonces la variable auxiliar y_{ij} vale 1. Cada instancia genera $n(n+k)$ variables[1] y $n(2k+1)$ *restricciones*[2] Así

[1] n^2 variables de y_{ij} y $n*k$ variables de x_{is}.

[2] n restricciones tipo 1, y $n*k$ restricciones del tipo 2, más un número igual de las tipo 3.

por ejemplo, al tener un problema con 15 vértices y 5 colores, tenemos 300 variables con 165 restricciones, lo que lo convierte en un problema binario grande. Para un problema de 60 vértices y 20 colores tenemos ¡4800 variables binarias y 2460 restricciones!

Desempeño del Modelo. Este modelo encuentra el valor mínimo óptimo de rigidez en instancias de 20 o menos vértices, llegando a necesitarse varias horas en encontrarle con una computadora actual, así que para resolver instancias grandes es necesario el uso de técnicas heurísticas de solución.

Algoritmo Genético-Glotón.

Descripción de la Técnica. Un algoritmo Genético (AG) es una Técnica Heurística que aplica los principios de la biología evolutiva en la ciencia de la computación. Los AG usan conceptos derivados de la biología tales como herencia, mutación, selección natural y recombinación, y son un caso particular de los algoritmos evolutivos. El inventor de esta técnica es John Holland, que a partir de ideas surgidas en 1962, en 1975 publica un libro llamado *Adaptation in Natural and Artificial Systems*, donde la técnica es descrita en su forma actual. En este libro se presenta el concepto de sistemas digitales adaptativos que, a partir de una cierta población inicial, utilizan la mutación, selección natural y recombinación, simulando un proceso de la evolución biológica, como una estrategia para resolver problemas.

Los algoritmos genéticos son conocidos por producir buenos resultados en algunos problemas, uno de los cuales es el problema de coloración mínima. Una característica que suelen tener las implementaciones de AG en general, es que producen rápidamente soluciones factibles y con calidad, el problema es cuando con un AG se buscan soluciones de mayor calidad: en este caso son relativamente lentos, comparados con otros métodos.

Entre las fortalezas de los AG se encuentran que son intrínsecamente paralelos (es decir, que se pueden ejecutar en varias máquinas o en varios procesadores al mismo tiempo), se desempeñan bien en problemas donde la función es compleja (es decir, es discontinua, ruidosa, cambia con el tiempo o tiene muchos óptimos locales). Finalmente una última ventaja es que se comporta como un "relojero ciego": ya que las decisiones están basadas en la aleatoriedad, todas las posibles líneas de búsqueda están abiertas a un AG. Por el contrario, una estrategia de búsqueda con base en conocimiento previo, empieza eliminando muchos caminos a priori, dejando de lado cualquier solución innovadora que se pudiera presentar ahí. Así los AG carecen de preconcepciones de "como se deben hacer las cosas" o sobre "lo que no se debe de hacer". Esta última característica la comparten todos los algoritmos evolutivos: no les importa si una solución va en contra de creencias establecidas: lo único que importa es que la solución funcione.

Descripción del Algoritmo. El Algoritmo Genético para el PCR fue propuesto en [Yánez, 2003]. Posiblemente fue la primera técnica heurística que se intentó, debido a los buenos

resultados que han arrojado los algoritmos genéticos en el problema de coloración mínima (que como se ha mencionado es un caso especial ya muy trabajado del PCR).

En este algoritmo los autores generaron mediante un glotón una población inicial de 20 elementos; a cada elemento de esta población le asignaron una probabilidad de cruza con cualquier otro elemento inversamente proporcional a su rigidez y al número de aristas mal coloreadas. Bajo este criterio se eligen dos elementos de acuerdo a esta probabilidad relativa y se cruzan entre ellos; cada elemento tiene una probabilidad de cruza en cada generación de 60%. Después a cada elemento se le asignó una probabilidad entre 0.05 o 0.1 de sufrir una mutación[3]. Este proceso se realizo durante 50 generaciones. El algoritmo en pseudo código se muestra a continuación:

Algoritmo Genético Glotón para la Coloración Robusta

```
INICIO {Programa}
    Se genera una población inicial de tamaño 20 con una técnica glotona.
    Se define la función de aptitud = rigidez + vértices mal coloreados.
    DESDE m=1 a 50
        Para el 60% de la población realiza:
            Elegir dos elementos de la población de acuerdo a su aptitud.
            Cruzar los padres elegidos para obtener dos hijos.
            SI alguno o ambos hijos son mejores que alguno de sus padres,
              ENTONCES reemplazar el (los) padre(s) con menor función de aptitud
              por el (los) hijo(s).
        FIN {Para}
        Elegir algún elemento, dicho elemento mutará con una probabilidad de 0.05
            (o bien, 0.1)
        SI hay mejora con el mutante, ENTONCES la coloración mutante toma el
            lugar de la coloración original.
    FIN {Desde}
FIN {Programa}
```

Desempeño del Algoritmo. Con este algoritmo solo podemos alcanzar el valor óptimo para instancias pequeñas (unos 10 vértices). Aparentemente para instancias grandes su desempeño es malo, aunque su principal ventaja es que es un algoritmo muy rápido, incluso se la llegado a probar en instancias de 1000 vértices, aunque sin garantía alguna en la robustez de las soluciones.

Algoritmo de Enumeración Parcial.

Descripción del Algoritmo. En este algoritmo, propuesto también en [Ramírez, 2000], se busca encontrar los clanes de orden 2 y 3, en el grafo complementario, generando

[3] En [Ramirez, 2000] se menciona una probabilidad de mutación de 0.05, y en [Ramirez, 2003] se menciona una probabilidad de 0.1

subgrafos que se colorean válidamente. Este algoritmo se muestra el pseudo código simplificado:

Algoritmo de Enumeración Parcial para el Problema de Coloración Robusta

```
INICIO {Programa}
    Ordenar las aristas complementarias de acuerdo a su penalización.
    Inicializar las variables n3(n,k), n2(n,k,n3) y n1(n,n2,n3).
    MIENTRAS (n1 ≥ 0 y solucion ≠ 1)
        Determinar n3 clanes de orden 3
        Determinar n2 clanes de orden 2
        SI (existen n3 y n2 clanes) ENTONCES
            Solucion = 1
          SI NO
            n3 = n3 + 1; recalcular n2 y n1
        FIN {Si}
    FIN{Mientras}
FIN {Programa}
```

Desempeño del Algoritmo. Su comportamiento es similar, cuando no ligeramente superior, al algoritmo genético, pero tiene dos problemas: no es tan rápido como el algoritmo genético y nos limita a trabajar con coloraciones tales que cumplan $k \geq n/3$ siendo que es posible encontrar coloraciones válidas con valores menores de k.

Algoritmo de Búsqueda Tabú.

Descripción de la técnica. La Búsqueda Tabú (BT) propuesta por Fred Glover en 1986, es una metaheurística iterativa que explora un conjunto de soluciones dentro de una cierta vecindad. Esta técnica mejora el desempeño de una búsqueda local utilizando estructuras de memoria. Con la finalidad de salir de óptimos locales se hace esta búsqueda prohibiendo (haciendo "tabú") los últimos movimientos durante una cierta cantidad de iteraciones. A veces, cuando se ha encontrado una solución mejor que todas las anteriores se acepta ésta aunque esté catalogada como Tabú; a este mecanismo se le llama "criterio de aspiración". Además de la memoria de últimos movimientos (llamada Memoria a Corto Plazo o MCP) se suele llevar un registro de todos los movimientos de mejora realizados (esta es llamada Memoria a Largo Plazo o MLP) a partir de la cual se puede deducir que combinaciones probablemente harán una mejora o bien cuales opciones no se han intentado.

Esta técnica es de las que mejor fama gozan dentro del mundo de los problemas combinatorios: su universo de aplicación abarca los sistemas complejos, combinatorios y no lineales entre muchos otros. Un problema de los algoritmos de BT es que suele ser

difícil de programar y de darle mantenimiento, especialmente al generar un algoritmo con memoria a largo plazo.

Descripción del Algoritmo. El Algoritmo Tabú para PCR fue propuesto por Ramírez, Gutiérrez, López y Yánez en [Ramírez, 2003], solamente considera una MCP y está acotado por un cierto número de iteraciones de búsqueda, el cual se pide como variable al inicio de cada corrida. A continuación se bosqueja:

Algoritmo de Búsqueda Tabú para el Problema de Coloración Robusta

```
INICIO {Programa}
        Generar una solución inicial (Método Glotón)
        DESDE n=1 hasta Número_de_Iteraciones
                Analiza las soluciones vecinas a un cierto vértice elegido aleatoriamente.
                Se toma la mejor entre las soluciones permitidas.
                Si una solución tabú es mejor que cualquiera anterior, se acepta.
                Se actualiza la MCP
        FIN {Desde}
FIN {Programa}
```

Desempeño del Algoritmo. Este algoritmo tiene un desempeño superior a los algoritmos Genético y Enumeración Parcial, antes descritos, dando por lo general soluciones tan buenas como el Algoritmo Exacto, aunque en ocasiones las soluciones solo son cercanas al optimo, pero se obtienen en cuestión de segundos, y no en horas, como pasa con el algoritmo exacto.

Algoritmo de Recocido Simulado.

Descripción de la Técnica. La técnica de Recocido Simulado (RS), fue propuesta por Kirkpatrick, Gelatt y Vechi en 1983[4], toma su nombre e inspiración del concepto de recocido en metalurgia. La técnica de recocido involucra el calentamiento y el enfriamiento controlado de un material para incrementar el tamaño de los cristales y reducir sus defectos. El calor causa que los átomos se desliguen de sus posiciones iniciales (un mínimo local de la energía interna) y emigren aleatoriamente a través de estados de mayor energía. El lento enfriado da más oportunidades de encontrar configuraciones con menor energía interna que el valor inicial.

En la implementación del algoritmo de RS, lo que buscamos es minimizar una función H, a la cual se asocia un costo a cada estado del sistema, llamado "temperatura" T. El algoritmo iterativamente propone cambios y dichos cambios son aceptados o rechazados por el Criterio Metrópolis: Si la función costo disminuye ($\Delta H < 0$) la nueva solución es simplemente aceptada. En otro caso, se acepta con una probabilidad de $\exp(-\Delta H / T)$.

[4] S. Kirkpatrick and C. D. Gelatt and M. P. Vecchi, *Optimization by Simulated Annealing*, Science, Vol. 220, Number 4598, pages 671-680, 1983. Una referencia muy completa sobre RS es [Gutiérrez, 1991].

Por cada iteración, la temperatura se disminuye en un factor β (por ejemplo, entre 0.9 o 0.99) y el número de soluciones S evaluadas por iteración es aumentado en un factor α.

Descripción del Algoritmo. El algoritmo de Recocido Simulado para Coloración Robusta fue propuesto en [Ramírez, 2003]. El algoritmo se divide en dos fases: una fase inicial se busca una *k-coloración* válida y en una segunda fase se busca disminuir la rigidez. El pseudo código del algoritmo se presenta a continuación:

Algoritmo de Recocido Simulado para el PCR

INICIO
  Encontrar una coloración válida mediante una búsqueda local.
  Se genera una función *H = rigidez + μ*(vértices mal coloreados)*
  Se determina un a temperatura inicial de $T=\sqrt{n}$. $\beta \approx 0.95$ y $\alpha > 1$
  MIENTRAS *T > "Temperatura mínima"*
    Si $\Delta H < 0$ se acepta la solución
    Si no, generamos un número aleatorio en el rango factible. Si $\exp(-\Delta H / T)$ es mayor que dicho número, se acepta la solución.
    Se ajustan los parámetros: $T = T * \beta$; $S = S * \alpha$
  FIN {Mientras}
FIN {Principal}

Desempeño del Algoritmo. De los métodos utilizados para resolver las instancias propuestas en [Ramírez, 2003], Recocido Simulado este es el que mejores resultados da, con muy buenos tiempos de ejecución.

Cotas para Coloraciones Robustas.

En [Ramírez, 2005] actualmente se desarrolla un algoritmo que obtiene cotas superior e inferior para cualquier instancia, el cual se basa en encontrar los clanes del grafo complementario, como en el algoritmo de Enumeración Parcial.

Algoritmo de Cotas para el PCR

INICIO
  Solución = 0
  Ordenar las aristas complementarias por penalización creciente.
  SI $3c \geq n$ ENTONCES
  $n_3 \geq n - 2c$
  $n_2 = \max\{n - c - 2n_3; 0\}$
  $n_1 = n - 2n_2 - 3n_3$
  MIENTRAS $n_1 \geq 0$ Y Solución ≠ 1
    Determinar n_3 clanes de orden 3

Determinar n_2 clanes de orden 2
SI (existen n_3 y n_2 clanes) ENTONCES
Solucion = 1
SI NO
$n_3 = n_3 + 1$; recalcular n_2 y n_1
FIN {Si}
FIN{Mientras}
SI solución=0 no se pudo encontrar una coloración válida. Incrementar el orden de los clanes
Si solución=1 ENTONCES
La **cota inferior** de rigidez se obtiene sumando las penalizaciones de las primeras $3n_3 + 2n_2$ aristas de la lista ordenada, aún si estas no forman 2-clanes o 3-clanes.
La **cota superior** de rigidez se obtiene sumando las penalizaciones de las últimas $3n_3 + 2n_2$ aristas de la lista ordenada, aún si estas no forman 2-clanes o 3-clanes.
FIN {Principal}

La rapidez de este algoritmo es asombrosa: para una instancia de más de 200 vértices, tarda menos de un segundo en encontrar las cotas de rigidez. Pero la utilidad real de este algoritmo, en su versión actual, esta acotada al tamaño de la instancia: mientras menor es la instancia, su eficiencia es mayor. Por ejemplo, para una cierta instancia de 10 vértices, en la cual se aplicó la técnica, el intervalo en el que esperamos encontrar el valor mínimo de rigidez es (2.4, 4.24), siendo el óptimo 4.2386, la cual es una excelente cota dada su rapidez. En cambio, cuando calculamos las cotas para una instancia de 120 vértices, el intervalo en que esperamos la rigidez es (1.9, 27.5) lo que es un intervalo grande para la menor rigidez conocida para esta instancia específica, es de 11.3507.

Así, en su versión actual, este generador de cotas es muy eficiente para instancias con unos 60 vértices a lo más.

2.2. Sumario de resultados obtenidos con los métodos de solución descritos.

En la *tabla 2.1.* se muestra un comparativo de resultados, en instancias idénticas, para los algoritmos descritos en este capítulo[5]. Las instancias son grafos aleatorios entre 10 y 30 vértices con una densidad de 0.5, y para el grafo complementario, la rigidez es un número aleatorio con distribución uniforme entre *0* y *1*. Se consideró en cada instancia una coloración para k y *k+1*. El asterisco indica que la solución es un valor óptimo y los valores en **negritas** indican las mejores soluciones factibles (de menor rigidez) encontradas en el problema.

[5] A excepción del algoritmo genético, cuyo desempeño es muy similar al algoritmo de Enumeración Parcial

Como se mencionaba antes, hasta instancias con menos de 20 vértices, el modelo exacto de solución puede encontrar el óptimo en un tiempo razonable. El algoritmo de Enumeración Parcial propuesto también en [Ramírez, 2000] se comporta muy bien con instancias pequeñas, pero jamás llega a ser tan bueno como el Modelo Exacto. Las cotas generadas por el algoritmo propuesto en [Ramírez, 2005] paulatinamente empiezan a dejar de ser eficientes, pero las cotas son útiles para instancias medianas.

Tabla 2.1. Rigidez encontrada por los diferentes métodos [Ramírez, 2003] y [Ramírez, 2005]

Instancia			***Rigidez en una Solución Válida***				
$G_{n,0.5}$	n	k	*Enumeración Parcial*	*Búsqueda Tabú*	*Recocido Simulado*	*Modelo 0-1*	Cotas Superior e Inferior
al(10)	*10*	*4*	**4.2386***	**4.2386***	**4.2386***	**4.2386***	*2.41 – 4.24*
		5	**1.8716***	**1.8716***	**1.8716***	**1.8716***	*1.03 – 1.87*
al(11)	*11*	*4*	**3.4450***	**3.4450***	**3.4450***	**3.4450***	*2.28 – 3.45*
		5	*2.4988*	**2.0227***	**2.0227***	**2.0227***	*1.81 – 2.50*
al(12)	*12*	*4*	**5.0815***	**5.0815***	**5.0815***	**5.0815***	*3.29 – 5.08*
		5	*3.8561*	**2.9041***	**2.9041***	**2.9041***	*2.86 – 3.86*
al(13)	*13*	*5*	*4.8958*	**4.0440***	**4.0440***	**4.0440***	*1.93 – 4.89*
		6	*2.7845*	**2.2765***	**2.2765***	**2.2765***	*1.67 – 2.78*
al(14)	*14*	*5*	*6.4479*	*5.3659*	*5.9756*	**5.2836***	*2.15 – 6.45*
		6	*3.7227*	**2.9387***	**2.9387***	**2.9387***	*1.41 – 3.72*
al(15)	*15*	*5*	*8.3658*	*6.0683*	**5.6376***	**5.6376***	*2.54 – 8.37*
		6	*5.0757*	**3.5592***	**3.5592***	**3.5592***	*1.79 – 5.08*
al(20)	*20*	*7*	*8.6477*	*6.0236*	**4.9557**	---	*2.47 – 8.65*
		8	*5.3147*	*3.9421*	**3.2285**	---	*1.89 – 5.31*
al(25)	*25*	*9*	*7.0819*	*6.7866*	**5.5344**	---	*1.87 – 7.08*
		10	*5.5987*	*4.8113*	**3.7923**	---	*1.43 – 5.60*
al(30)	*30*	*10*	*10.5818*	*10.1346*	**7.6417**	---	*2.42 –10.58*
		11	*8.5675*	*7.4842*	**4.7623**	---	*1.97 – 8.57*

Con el Algoritmo de Búsqueda Tabú se obtuvieron siempre coloraciones con menor rigidez, respecto a las obtenidas con el Algoritmo de Enumeración Parcial.

Pero sin duda alguna, de los algoritmos desarrollados previos a este trabajo de investigación, el mejor fue el algoritmo de Recocido Simulado. Se encontró el valor óptimo en todos excepto en una ocasión, y en los valores donde el algoritmo Exacto no era útil, el de Recocido Simulado siempre encuentra la solución con menor rigidez.

Éste es el Estado del Arte del Problema de Coloración Robusta. El próximo capítulo explica en qué consisten las Técnicas Búsqueda Dispersa y GRASP, cuya implementación al Problema de Coloración Robusta es el objetivo de esta tesis.

Capítulo 3. Las Técnicas Búsqueda Dispersa y GRASP

Las meta-heurísticas han tenido un enorme desarrollo en los últimos treinta años, y se han utilizado ampliamente en problemas "difíciles" de optimización: desde modelos no lineales a entrenamiento de redes neuronales pasando por sistemas complejos, sin olvidar los problemas combinatorios. Tenemos las meta-heurísticas "clásicas": Recocido Simulado, Genéticos, y Búsqueda Tabú, a las cuales en tiempos recientes se han integrado las técnicas GRASP y Búsqueda Dispersa.

La técnica GRASP es una meta-heurística muy rápida que genera soluciones de alta calidad, la cual se divide en dos fases: una de construcción de soluciones, que utiliza para ello un algoritmo glotón aleatorizado y una fase de mejora, la cual realiza una búsqueda local. Este proceso se repite varias veces y la mejor solución encontrada se toma como el resultado.

La Búsqueda Dispersa es una técnica heurística evolutiva que se ha utilizado exitosamente en diversas aplicaciones, principalmente en modelos combinatorios, no lineales, simulación y sistemas complejos; Esta técnica se basa en integrar la combinación de soluciones con un método de mejora. La técnica se divide en 5 partes: primero, un Generador de Soluciones Iniciales (las primeras soluciones sobre las cuales se generarán todas las demás); segundo, un Método de Mejora (mediante el cual una solución es mejorada hasta alcanzar un óptimo local); tercero, un Método de Actualización del RefSet (de las mejores soluciones iniciales perfeccionadas generamos el primer RefSet, dicho conjunto ha eliminado las soluciones redundantes y se ha asegurado que exista una cierta diversidad); cuarto, un Método de Generación de Subconjuntos (generar un subconjunto de soluciones para producir nuevas); y quinto, un Método de Combinación de Soluciones (que transforma el conjunto de soluciones generadas en la cuarta parte, dentro de vectores de solución combinados). En este último punto es muy útil el llamado Religado de Trayectorias (PR: Path Relinking) Estas soluciones mezcladas son perfeccionadas mediante el llamado Método de Mejora; las nuevas soluciones que sean mejores que las actuales, se utilizan para generar un RefSet de segunda generación, el cual es usado para hacer nuevas soluciones, y así sucesivamente.

En este capítulo se detalla el uso de ambas técnicas aplicadas a problemas combinatorios. En la sección 3.1. se describe la Técnica GRASP. En la sección 3.2. se describe la Técnica de Búsqueda Dispersa. En la sección 3.3 se propone utilizar un GRASP para generar el Pool inicial.

3.1. La Técnica GRASP

[Feo, 1995], [Resende[a], 2003], [Aiex, 2002]

La técnica GRASP [Feo, 1995] parte de una solución inicial glotona aleatorizada, la cual es mejorada, por lo común mediante una búsqueda local. Esta implementación es la básica, sobre este esquema podemos hacer mejoras: La solución inicial se puede obtener, por ejemplo, mediante un algoritmo glotón sin aleatorizar[1] y el método de mejora podría hacerse implementando un recocido simulado. Como es de esperarse, mientras más complicado sea el algoritmo, mayor es la calidad de las soluciones, pero el tiempo de ejecución también se incrementa, llegando a un punto donde si se desea mayor calidad de soluciones, es más conveniente implementar un algoritmo en una técnica más compleja.

Para analizar la calidad de las soluciones que arroja esta técnica se puede ver como una técnica de muestreo repetitivo. Cada iteración del GRASP produce un valor de una distribución probabilística de todos los resultados posibles. La media y la varianza de su distribución son un indicativo de la calidad de resultados que se obtendrán utilizando esta técnica, por lo común se toman unas 100 "muestras" y se considera a la mejor de ellas como el resultado de la técnica.

Una característica especialmente atractiva de esta técnica es la gran facilidad con que puede ser implementada. Pocos parámetros deben ser supervisados (en su forma más elemental, solamente el número de iteraciones GRASP y tamaño de la lista de candidatos). Así, el desarrollo se puede enfocar en implementar estructuras de datos eficientes para asegurar iteraciones GRASP muy rápidas.

A continuación se detallan las dos fases de esta técnica:

1. ***Fase de construcción.*** Se utiliza un algoritmo Glotón aleatorizado. Primero expliquemos qué es un algoritmo glotón. Los glotones tienen tres características: es incremental (vamos agregando valores a una cierta función a cada paso); no tiene marcha atrás (una vez tomada una decisión no se puede modificar) y siempre se toma el mejor siguiente elemento de una lista de posibles soluciones. Como ejemplo de un algoritmo glotón tenemos el de "vecino más cercano", para resolver el Problema del Agente Viajero; el cual consiste en elegir alguna ciudad al azar y visitamos la ciudad que esté más cerca a la actual, de esa segunda ciudad tomamos la siguiente ciudad más cercana (sin considerar la primera, para así evitar un circuito). Continuamos este proceso hasta agotar las ciudades. Como vemos, este algoritmo glotón cumple con las tres características: vamos incrementando a cada paso la distancia recorrida para aumentar el recorrido parcial; no hay marcha atrás porque una vez seleccionada una ciudad no se puede retirar; y el criterio de selección es siempre tomar la distancia más pequeña disponible. Este algoritmo es extremadamente sencillo, y con él podemos obtener n trayectorias para un agente viajero de n ciudades las cuales están listas para aplicarles el método de mejora.

[1] Una técnica glotona sin aleatorizar solo se recomienda cuando el número de soluciones se incrementa al menos como el cuadrado del tamaño de la instancia.

En un Algoritmo Glotón *Aleatorizado*, no necesariamente consideramos la mejor siguiente opción para incluirla al conjunto solución, sino se considera a todos aquellos "candidatos" *c(e)*, que se encuentran dentro de un porcentaje α ($0 \leq \alpha \leq 1$) entre el mejor elemento, c_* y el peor de los candidatos posibles, c^*:

$$c(e) \leq c_* + \alpha(c^* - c_*)$$

El valor α es una especie de "porcentaje de tolerancia" donde podemos aceptar no solo al mejor, sino también a los candidatos más cercanos a dicho valor mínimo. Valores usuales de α son *0.1* o *0.2*, pero nada nos impide el uso de cualquier valor, incluyendo los valores extremos, $\alpha=0$ (algoritmo glotón estándar) ó $\alpha=1$ (solución totalmente al azar).

Por ejemplo, consideremos el PAV resuelto con un glotón de vecino más cercano aleatorizado. Si actualmente nos encontramos en una ciudad llamada "Fresno" y las opciones que nos quedan de ciudades por visitar, con sus respectivas distancias son:

Monterey – 190 Km.	*San Francisco – 261 Km.*
San José – 199 Km.	*Sacramento – 252 Km.*
Los Gatos – 204 Km.	*Santa Rosa – 319 Km.*

En este paso, tenemos que $c_* = 190$ y $c^* = 319$. Para $\alpha=0.1$ nuestra cota de inclusión a la lista es de $c(e) \leq 192+0.1(319-192) = 202.9$ *kms*, lo que incluye en la lista de candidatos a *Monterey* y *San José*. Si consideramos $\alpha=0.2$ entonces $c(e) \leq 215.8$ *kms*, en este caso la lista de candidatos la forman *Monterey, San José* y *Los Gatos*, y podemos incluir en el recorrido a cualquiera de ellos.

La principal ventaja de utilizar un glotón aleatorizado respecto a tomar los extremos de α igual a cero o uno, es la diversidad que ofrece: por ejemplo en el glotón del PAV propuesto, un glotón estándar encuentra las mejores soluciones, pero a lo más hay n soluciones posibles, por otro lado, usando una solución totalmente azarosa, nos da $n!$ soluciones, pero prácticamente todas de mala calidad. El glotón aleatorizado es un equilibrio entre calidad y diversidad.

2. ***Fase de Mejora.*** El método de mejora más utilizado es la búsqueda local, la cual consiste en explorar exhaustivamente[2] la vecindad de una solución en busca de una mejor; si no existe dicha solución mejorada en el vecindario, nos encontramos en un óptimo local.

   La búsqueda local es muy importante dentro del GRASP ya que nos permite hacer búsquedas de mejora dentro de regiones prometedoras del espacio de soluciones. En nuestro ejemplo del agente viajero, podemos tomar el "vecindario" de muchas formas, una de ellas es tomando todas las permutaciones entre dos ciudades, es decir, se intercambian los lugares en el vector de trayectoria, por ejemplo, si

[2] Para el caso de vecindarios grandes (por ejemplo, que nuestro vecindario crece proporcionalmente a una n^2 o más), después de un cierto valor de n, se puede considerar solo hacer un muestreo del vecindario.

tenemos la solución de PAV de 4 ciudades *(a,b,c,d)* un vecino de esta solución es *(a,c,b,d)* al intercambiar los lugares de las ciudades *b* y *c*. Para este vecindario tenemos *n(n-1)/2* vecinos posibles, en nuestro PAV de 4 ciudades tenemos 6 vecinos: *(b,a,c,d)*, *(c,b,a,d)*, *(d,b,c,a)*, *(a,c,b,d)*, *(a,d,c,b)* y *(a,b,d,c)*. si alguno de estos vecinos da una solución mejor, lo tomamos como la nueva solución y repetimos el proceso, hasta que ya no se puede lograr una mejora.

Pseudo Código de un GRASP Genérico.

INICIO {Programa}
DESDE *m=1* hasta *Tamaño_de_Muestra*

INICIO {Fase de Construcción}
- Generar una lista de candidatos para ser incluidos a la solución y ordenarlos de mejor opción a la peor.
- Tomar mediante algún criterio (los *k* mejores, una probabilidad de inclusión relacionada con su función de desempeño, etc.) uno de los mejores candidatos.
- Dado que el (los) nuevo(s) elementos ya se ha(n) incluido, calcular la nueva función costo.
- Repetir los pasos anteriores hasta llenar la solución.

FIN {Fase de Construcción}

INICIO {Fase de Mejora}
- Generar una "vecindad de soluciones" (cambio de una variable *0* a *1* o viceversa, permutar dos elementos, etc.)
- Probar las soluciones de la vecindad hasta encontrar una solución mejorada.
- Con centro en la nueva vecindad repetir el procedimiento
- Continuar hasta no encontrar una mejor solución.

FIN {Fase de Mejora}

FIN {Desde}
FIN {Programa}

3.2. La Técnica de Búsqueda Dispersa.

[Glover, 1998], [Martí, 2003], [Laguna, 2002]

La Búsqueda Dispersa *(SS: Scatter Search)*, junto con su generalización, el religado de trayectorias *(PR: Path Relinking)* se clasifican como algoritmos evolutivos ya que construyen nuevas soluciones a partir de combinar soluciones anteriores y se fundamenta en trabajos de Fred Glover hechos entre 1963 y 1965[3], pero es hasta [Glover, 1998] cuando toma su forma definitiva.

[3] F. Glover *Parametric Combinations of Local Job Shop Rules.* Chapter IV, ONR Research Memorandum no. 117, GSIA, Carnegie Mellon University, Pittsburg PA, 1963 y F. Glover *A Multiphase Dual Algorithm for the Zero-One Integer Programming Problem.* Operations Research, Vol. 13, No. 6-879, 1965

La Búsqueda Dispersa opera sobre un conjunto de soluciones, llamado Conjunto de Referencia (*RefSet*). Se combinan dos o más soluciones del *RefSet,* buscando crear nuevas soluciones que sean mejores que las originales. Al contrario de otros algoritmos evolutivos (como por ejemplo, los genéticos), el conjunto "población" suele ser pequeño (no más de 20 soluciones, por lo común se incluyen únicamente unas 10). Se basa en el principio de que la calidad de las soluciones puede mejorar mediante la combinación de dos o más de ellas en lugar de aisladamente. Otro aspecto de esta técnica es que integra la combinación de soluciones con la búsqueda local, lo que permite aprovechar a cada solución no solo por sí misma sino representar a todo un vecindario.

La Técnica de Religado de Trayectorias[4] es un refinamiento de los mecanismos utilizados en la técnica de Búsqueda Dispersa. En lugar de producir directamente una solución mezclando dos o más de ellas, el religado de trayectorias genera recorridos entre las soluciones seleccionadas y su espacio solución circundante. Esto es, al empezar por una cierta solución A, progresivamente es transformada en otra solución B. En el Método de Combinación de Soluciones podemos hacer la búsqueda entre dichas soluciones y en cada paso generar una solución parcial, a la cual se le puede aplicar el método de mejora y considerar su inclusión al Conjunto de Referencia.

Hay tres principios en que se sustenta la Metodología de Búsqueda Dispersa:

- En una elite muy pequeña de soluciones (en la mayoría de las aplicaciones con no más de 20 elementos), podemos almacenar las características (o la localización) de las soluciones óptimas.

- Se deben proporcionar mecanismos que exploren nuevas soluciones más allá de una solución óptima local. Con esto en mente, también es importante incorporar procesos heurísticos para mezclar soluciones y generar nuevas a partir de resultados anteriores.

- El uso de varias soluciones simultáneamente como parte central para generar nuevas combinaciones, incrementa las oportunidades de explotar la información contenida en la unión de soluciones de élite.

Elementos de la Búsqueda Dispersa
[Glover, 1998], [Martí, 2003]

En 1997 Fred Glover escribe su lista de instrucciones genéricas para Búsqueda Dispersa, [Glover, 1998]; este trabajo ha sido piedra fundamental en la construcción y difusión de la técnica de Búsqueda Dispersa.

Se pueden distinguir 5 etapas o elementos en la técnica:

1. *Un Generador de Nuevas Soluciones.* Un Generador de Nuevas Soluciones crea una colección de soluciones de prueba (o iniciales) y con las mejores de ellas, formar el primer *RefSet.* Estas soluciones se pueden producir aleatoriamente o

[4] Conocida en España también como re-encadenamiento de Trayectorias

usando una regla determinística para generar nuevas soluciones. Dicho conjunto de soluciones sobre las cuales vamos a hacer una selección de los mejores elementos, se le llama el conjunto grande o *BigSet o Pool*, y suele estar formado de unas 100 soluciones. Con base en las mejores soluciones de este conjunto, generamos el subconjunto de élite, llamado también Conjunto de Referencia o *RefSet.*

2. *Un Método de Mejora.* Este método transforma una solución de prueba a una o más soluciones perfeccionadas. Si al aplicar el Método de Mejora, la solución transformada no es de mayor calidad, simplemente usamos la solución original. El Método debe de ser rápido y eficiente (búsqueda local es lo más utilizado). En caso de que nuestro método de mejora sea muy bueno pero lento, por ejemplo, si utilizamos un Tabú o Recocido Simulado, podemos aplicarlo únicamente en las soluciones más prometedoras.

3. *Un Método de actualización del Conjunto de Referencia.* El *RefSet* debe de recibir mantenimiento después de cada generación, seleccionando las mejores soluciones, eliminando los elementos redundantes y preservando algo de diversidad, todo esto dentro de 10 o 20 elementos. Se recomienda que los elementos dentro del *RefSet* se ordenen de mayor a menor de acuerdo a su calidad. Para un *RefSet* genérico se recomienda una proporción de *50/50* entre calidad y diversidad, pero este criterio puede variar de acuerdo a las características de cada problema específico.

4. *Un Método de Generación de Subconjuntos.* Para generar nuevas soluciones a partir del conjunto de referencia, en esta paso se crean subconjuntos con sus elementos como base para nuevas soluciones combinadas, de esta forma especificamos cómo se tomarán los elementos para el Método de Combinación. La más simple de estas particiones es tomar todos los pares de elementos posibles (todos los subconjuntos de tamaño 2), esto significa $n(n-1)/2$ subconjuntos.

5. *Un Método de Combinación de Soluciones.* En este paso, transformamos el grupo de soluciones producidas por el Método de Generación de Subconjuntos en uno o más vectores solución combinados y exploramos exhaustivamente todos los subconjuntos elegidos. Esta es la forma en que las soluciones viejas se mezclarán para crear nuevas. Una vez que mejoremos estas soluciones tenemos dos opciones: Agregar inmediatamente un elemento que sea mejor que el peor elemento del *RefSet* (actualización dinámica) o reconstruir el *RefSet* al final de cada generación (actualización estática).

La técnica de religado de trayectorias, es considerada como una extensión del Método de Combinación de Soluciones. En el religado a partir de una solución inicial, progresivamente nos movemos hacia (agregamos elementos desde) una solución guía hasta alcanzarla; a cada paso mejoramos la solución mezclada, buscando óptimos locales a lo largo de la trayectoria. Si encontramos otra solución de alta calidad, en el futuro será usada como ancla para nuevas búsquedas.

A continuación mostramos un diagrama que muestra los elementos de la Búsqueda Dispersa en acción.

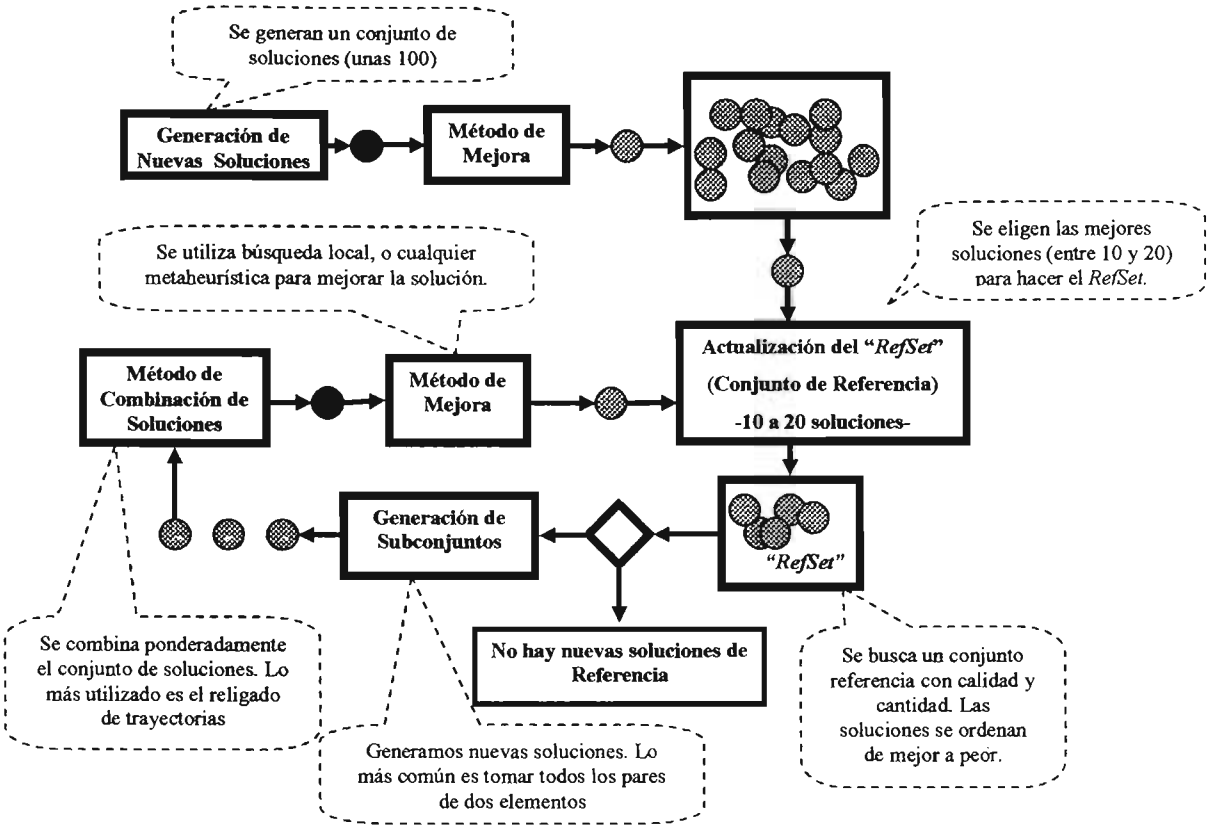


Figura 3.1. Diagrama de la técnica de Búsqueda Dispersa (Referencia: [Martí, 2003])

Pseudo Código de una Búsqueda Dispersa Genérica.

INICIO {Programa}
DESDE *i=1* a *Tamaño_BigSet*
Crear una solución con el Generador de Nuevas Soluciones.
Usar en esa solución el Método de Mejora.
Agregar la solución al *BigSet*
FIN {Desde i}

Con las mejores soluciones (preservando diversidad) construimos el conjunto de referencia inicial (proceso de actualización del Conjunto de Referencia).

MIENTRAS *NewSol=VERDADERO* ejecuta:
NewSol=FALSO
Utilizar el Método de Generación de Subconjuntos
PARA cada subconjunto creado:
Utilizar el Método de Combinación de Soluciones (Religado de Trayectorias para mejores resultados) para obtener una o más soluciones de prueba.
Aplicarles el Método de Mejora.
Aplicar el Método de Actualización del Conjunto de Referencia

```
        SI hay un Nuevo elemento en el RefSet ENTONCES NewSol=VERDADERO
      FIN {Para}
    FIN {Mientras}
FIN {Programa}
```

En la Técnica de Búsqueda Dispersa primero generamos un conjunto de soluciones iniciales diversas con el *Generador de Nuevas Soluciones*, las cuales se almacenarán en el *BigSet*. Los mejores elementos del *BigSet* formarán parte del primer *RefSet*. Una vez que hayamos decidido qué *Método de Generación de Subconjuntos* vamos a utilizar, en este subconjunto aplicamos el *Método de Combinación de Soluciones / Religado de Trayectorias* para generar nuevas soluciones y mejorarlas. Con las mejores soluciones obtenidas mediante este proceso, actualizamos y damos *mantenimiento* al *RefSet*, hasta que, una vez más, aplicamos el *Método de Generación de Subconjuntos* para empezar un nuevo *Religado de Trayectorias* y así sucesivamente.

Uso de las herramientas de Búsqueda Dispersa [Glover, 1998] y [Glover, 2000]

Para ejemplificar cómo se pueden utilizar estas herramientas en problemas combinatorios, Fred Glover hace uso de dos ejemplos específicos: uno en donde el vector de solución son variables tipo 0-1 y otro para problemas de permutación.

Generador de Nuevas Soluciones. Una forma elemental de garantizar que el conjunto inicial sea representativo del espacio solución de nuestra instancia, es realizando un muestreo aleatorio simple del mismo. Por ejemplo, para un problema binario podemos llenar el vector solución con valores al azar con probabilidad de 0.5 de ser cero o uno. Para el caso del Problema del Agente Viajero, iniciando con un vector cualquiera, digamos (1,2,3,4,5,6,7), podemos tomar cualquier entrada y permutarla con cualquier otra (incluso consigo misma), acto seguido tomamos la segunda entrada y la permutamos con cualquier otra de las entradas restantes (2 a 7) y así hasta agotar el vector. Se presenta un ejemplo para generar un recorrido al azar del agente viajero de 7 ciudades:

(1,2,3,4,5,6,7)	*Vector inicial*
(**5**,2,3,4,**1**,6,7)	*Generamos un número aleatorio entre 1 y 7, (supongamos que fue 5), así permutamos la primera y la quinta entrada.*
(5,**3**,**2**,4,1,6,7)	*Fijamos la primera entrada y permutamos la segunda con cualquier otra al azar, digamos la 3.*
(5,3,**6**,4,1,**2**,7)	*Permutamos la tercer entrada con cualquier restante (6?) y así sucesivamente*
(5,3,6,**7**,1,2,**4**)	
(5,3,6,7,**1**,2,4)	*La quinta entrada permutó consigo misma, quedando el vector igual.*
(5,3,6,7,1,4,2)	*Vector final*

Este último vector es un recorrido aleatorio listo para aplicarle el método de mejora.

En algunos casos es útil usar un método determinístico. En [Glover, 1998] se presenta como alternativa para el caso de problemas binarios, "prender o apagar" la *i-ésima* entrada del vector (si es uno, hacerlo cero, y si es cero, hacerlo uno); o bien, solo no cambiar el *i-ésimo* valor de un vector, por ejemplo, para el vector (0,0,...,0) surgirían los vectores (1,1,1,...,1) al cambiar los valores adyacentes, (1,0,1,0,1...) al cambiar uno si y otro no, (1,0,0,1,0,0,1,...) al cambiar uno si y dos no, etc. Al igual podríamos generar sus complementos (0,0,...,0), (0,1,0,1,0,...), (0,1,1,0,1,1,0,...) etc.

Mantenimiento y actualización del Conjunto de Referencia. En [Glover, 1998] se presenta un pseudo código genérico de cómo realizar el mantenimiento y actualización del *RefSet*. Cabe destacar dos variables que se mencionan: *E(x)*, que es el valor de la función objetivo para una solución *x* y la función *hash(x)*, la cual indica cuando dos soluciones son exactamente equivalentes (como [1,2,3,4,1] y [3,4,1,2,3] para un agente viajero de 4 ciudades) o cuando son parcialmente iguales (como serian [1,2,3,4,5,6,8,7] y [1,2,3,4,5,6,7,8]), se busca que no existan dos soluciones exactamente iguales y para mantener diversidad, evitar tener demasiadas soluciones con vectores *hash(x)* muy similares.

Método de Mejora. Se recomienda que la mejora sea una técnica sencilla: un glotón o búsqueda local. Fred Glover recomienda ampliamente el método genérico de "un movimiento" (*one-move technique*). En este enfoque se hace un solo cambio (un intercambio de lugares en una secuencia de ciudades del agente viajero, "prender o apagar" una variable binaria, etc.) y se prueba si hay mejora; si lo hace se toma la nueva solución como la representativa y se busca mejorarla una vez más. Así se continúa hasta que ya no se puede encontrar una mejor solución.

Escogiendo subconjuntos de las soluciones de referencia. Se fija la atención en un conjunto relativamente pequeño de subconjuntos con características útiles. Se consideran cuatro tipos de subconjuntos:

- ✓ Subconjunto Tipo 1: Todos los subconjuntos de dos elementos.
- ✓ Subconjunto Tipo 2: Los subconjuntos de tres elementos derivados de los subconjuntos de dos elementos + la mejor solución no incluida en el subconjunto.
- ✓ Subconjunto Tipo 3: Los subconjuntos de cuatro elementos derivados de los subconjuntos de tres elementos + la mejor solución no incluida en el subconjunto.
- ✓ Subconjunto Tipo 4: Los subconjuntos consistentes en los mejores i elementos, desde $i = 5$ a b.

Método de Combinación de Soluciones. El religado de trayectorias es el método más recomendado para mezclar las soluciones, Como un ejemplo de su uso, hagamos el religado de dos vectores binarios, del (0,0,0,0,0) al (1,1,1,1,1). Hay muchísimas formas de mezclarlas, una de ellas seria agregar elementos estocásticamente o secuencialmente mediante algún criterio. Se presentan a continuación tres caminos posibles para llegar de la solución inicial la final en 4 pasos: el primero muestra una inclusión secuencial trivial (considera solo el orden de los elementos del vector) y en el otro se consideran dos caminos de mezclado al azar:

	Inclusión Secuencial	Inclusión Aleatoria 1	Inclusión Aleatoria 2
Solución Inicial	(0,0,0,0,0)	(0,0,0,0,0)	(0,0,0,0,0)
x(1)	(1,0,0,0,0)	(0,0,0,0,1)	(0,0,1,0,0)
x(2)	(1,1,0,0,0)	(0,1,0,0,1)	(0,1,1,0,0)
x(3)	(1,1,1,0,0)	(0,1,0,1,1)	(0,1,1,0,1)
x(4)	(1,1,1,1,0)	(0,1,1,1,1)	(1,1,1,0,1)
Solución Final	(1,1,1,1,1)	(1,1,1,1,1)	(1,1,1,1,1)

3.3. GRASP para iniciar la Búsqueda Dispersa.

Debido a que un buen GRASP genera un grupo representativo de soluciones de alta calidad y con rapidez, una idea que ha tenido buena aceptación en fechas recientes es utilizar un GRASP para generar el conjunto inicial o *"BigSet"* del Algoritmo de Búsqueda Dispersa. El generador de nuevas soluciones se puede realizar con la fase 1 (glotón aleatorizado) del GRASP y la fase 2 (perfeccionamiento de soluciones) se puede realizar con el método de mejora general. El algoritmo modificado se muestra a continuación:

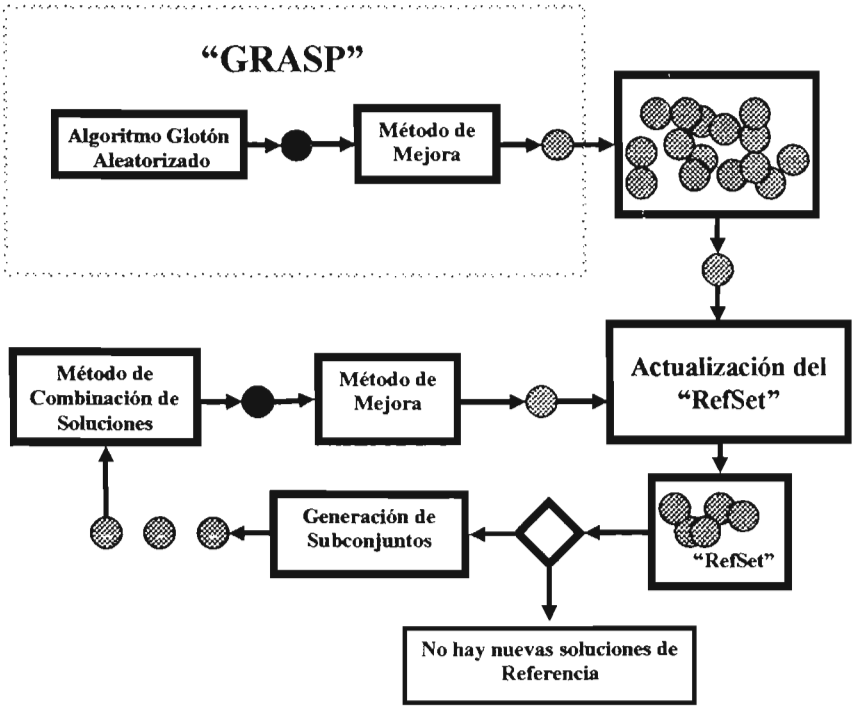


Figura 3.2. GRASP en la Búsqueda Dispersa

Ya hemos explicado en qué consiste el Problema de Coloración Robusta, sus técnicas de solución y en qué consisten las técnicas GRASP y Búsqueda Dispersa. El siguiente paso es explicar cual es la contribución de esta tesis, es decir, cómo se diseñaron los algoritmos, su implementación y los resultados obtenidos. Esto es lo que se hará en los próximos tres capítulos.

4. Elementos de un Algoritmo de Búsqueda Dispersa y un GRASP para el Problema de Coloración Robusta.

En el capítulo anterior se mostraron los elementos necesarios para hacer un Algoritmo GRASP y para uno de Búsqueda Dispersa. Dichos elementos se presentaron genéricamente, sin detalle alguno de su implementación en alguna aplicación. Estos procedimientos genéricos se deben "aterrizar" a un problema en particular, el Problema de Coloración Robusta en el caso de esta tesis.

En este capítulo los algoritmos genéricos son descompuestos en sus elementos y se optimizan por separado buscando el mejor desempeño posible dentro de cada subsistema. Para cada uno de los cinco elementos de la Búsqueda Dispersa, se explican las mejores opciones que se encontraron, los resultados de desempeño, análisis, y algunas recomendaciones. Se muestra los procedimientos en pseudo código. Para el GRASP buscamos alta calidad de las soluciones con un tiempo de ejecución mínimo, ya que se utilizará el GRASP para inicializar el Conjunto de Referencia del Algoritmo de Búsqueda Dispersa, por lo tanto deseamos las mejores soluciones posibles, con un costo en tiempo razonable.

Este capítulo se divide como sigue: La sección 4.1 y 4.2. se enfocan en el GRASP, donde la primera fase es utilizada como Generador Inicial de Soluciones y la segunda fase se utiliza como el Método de Mejora de la Búsqueda Dispersa, tal y como se sugiere al final del capítulo anterior. Las secciones 4.3. a 4.5 se enfocan en los otros tres elementos, que son característicos únicamente de la Búsqueda Dispersa: Método de Actualización del Conjunto de Referencia, Método de Generación de Subconjuntos y un Método de Combinación de Soluciones. La sección 4.6. presenta el pseudo-código de este algoritmo preliminar.

Plano general del Proyecto

Se seleccionó un GRASP para generar soluciones iniciales de calidad, con las mejores generamos el primer Conjunto de Referencia, al cual se le aplicará un ciclo de Búsqueda Dispersa, usando como método de mejora una Búsqueda Local.

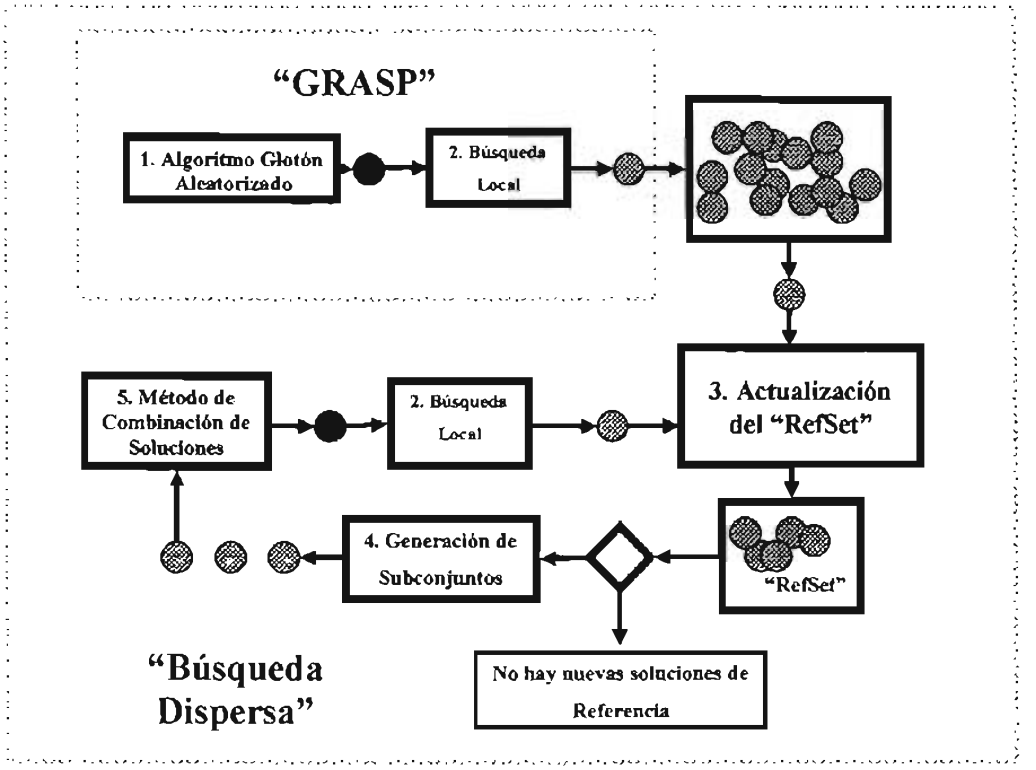


Figura 4.1. Los cinco elementos por mejorar dentro de los algoritmos.

Se propusieron, probaron y optimizaron cada uno de los elementos numerados y mostrados en la *figura 4.1.* En las 5 secciones de este capítulo se muestran las opciones más prometedoras.

Ambiente Computacional. Los experimentos se realizaron en una computadora Compaq Presario 5006LA (512MB RAM, procesador Pentium 4 a 1.2 GHz). Los algoritmos inicialmente se programaron en *Visual Basic 6.0*, pero su inestabilidad, lentitud y el ser un lenguaje fuertemente tipificado, lo volvieron una mala opción de programación. Por esta razón se emigró el código a *Power Basic for Windows 7.0*, el cual tiene un desempeño similar a *Fortran for Windows* o *C++*. El cambio de lenguaje de programación aumentó la velocidad de ejecución 20 veces. Los resultados de velocidad mostrados en este capítulo fueron los obtenidos con *Power Basic*.

Instancias de Prueba. Se utilizaron grafos aleatorios del tipo $G_{n.0.5}$ (es decir, grafos con n vértices y con densidad de aristas en el grafo original de 0.5). El número de vértices por grafo abarca los valores $n = 15, 20, 25 \ldots 120$. El número de colores utilizado por instancia fue el valor redondeado de $n/3$ como se utilizan en [Ramírez, 2003]. Se utilizo la misma

instancia por vértice para cada método de solución probado, iterando cien veces cada instancia.

4.1. Método de Generación de Soluciones.

Algoritmo Glotón Aleatorizado. Se consideraron dos técnicas mejoradas para producir las soluciones iniciales: la primera es un vector de coloración generado totalmente al azar, y después a esa solución se le aplica el Método de Mejora. El otro método encuentra una coloración válida con un método glotón y luego la solución es mejorada.

- **Solución Inicial Aleatoria Mejorada.** Elegimos un vector de coloración totalmente aleatorio (es decir, cada vértice tiene una probabilidad de $1/k$ de tener algún color). Podemos conseguir de esta forma k^n diferentes soluciones iniciales. A partir de esta solución, mejoramos la solución reduciendo el número de vértices mal coloreados mediante una búsqueda local (ver sección 4.2.). Esta solución aleatoria con mejora fue diseñada de tal forma que no favorece a ningún vértice ni color, lo cual nos permite generar un *RefSet* representativo del espacio de soluciones del grafo que estemos analizando.

  Descripción del Procedimiento y sus Variables. Generamos un vector aleatorio *Col(j)* el cual se mejora en una secuencia al azar de sus vértices *Sec(j)*. Probamos cada color sin reemplazo también al azar, con la variable *SeqCol(i)*. El proceso se repite hasta no encontrar mejora alguna.

  Pseudo código para la Solución Inicial Aleatoria Mejorada.

```
INICIO {Procedimiento}
    Construir una Coloración Inicial Aleatoria Col(j)
    MIENTRAS tengamos una mejora en la coloración:
       Generamos una secuencia de visita de vértices Sec(j)
       DESDE j=1 a n
            Generamos una secuencia de prueba de colores SeqCol(i)
            DESDE i=1 a k
                Si Col(Sec(j))=SeqCol(i) reduce el número de vértices mal
                coloreados, entonces reemplaza la solución
            FIN {Desde i}
        FIN {Desde j}
    FIN {Mientras}
FIN {Procedimiento}
```

- **Solución Inicial Glotona Mejorada.** El algoritmo elige cualquier vértice no pintado y le aplica un color al azar, si la coloración es válida se deja pintado con ese color y se pasa a otro vértice donde se repite el proceso hasta agotar los vértices. Si ningún color puede hacer una coloración válida, se pinta con un color al azar.

Una vez obtenida esta coloración glotona, Esta es perfeccionada con una búsqueda local (ver sección 4.2).

Pseudo Código de la Solución Inicial Glotona Mejorada

```
INICIO {Procedimiento}
  DESDE m=1 a BigSetSize
    Generar una secuencia en que los vértices serán "visitados": Sec(j)
    DESDE j=1 a n
        Generar una secuencia en que los colores serán probados SeqCol(i)
        DESDE i=1 a k
            Si Col(Sec(j)) = SeqCol(i) es una coloración válida, pintar ese vértice
               con ese color
        FIN {Desde i}
        Si Col(j) = ϕ , pintar el vértice con un color al azar.
    FIN {Desde j}

    MIENTRAS tengamos una mejora en la coloración:
        Generamos una secuencia de visita de vértices Sec(j)
        DESDE j=1 a n
            Generamos una secuencia de prueba de colores SeqCol(i)
            DESDE i=1 a k
                Si Col(Sec(j))=SeqCol(i) reduce el número de vértices mal
                coloreados, o la rigidez sin empeorar la coloración, entonces
                reemplaza la solución.
            FIN {Desde i}
        FIN {Desde j}
    FIN {Mientras}
  FIN {Desde m}
FIN {Programa}
```

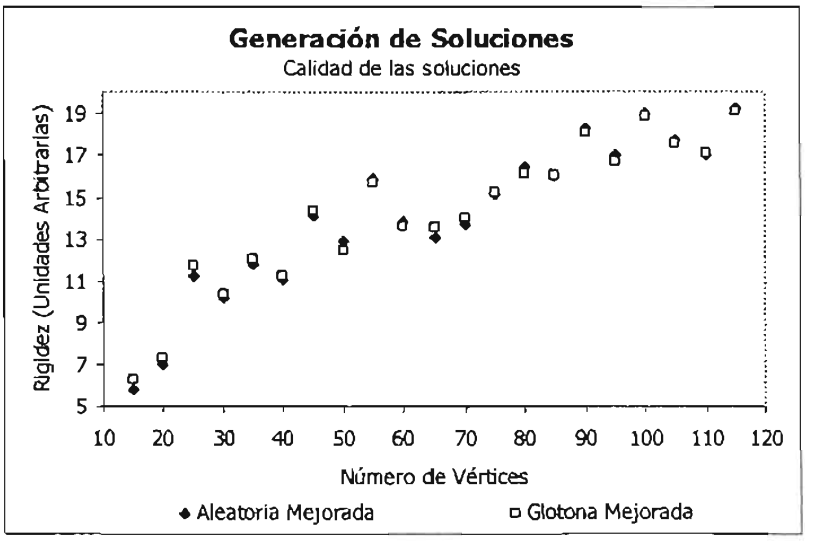
Pruebas Comparativas para los Métodos de Inicialización. Se tomaron en cuenta para medir su desempeño dos características: calidad de la solución y tiempo de ejecución del método.

- *Calidad de las Soluciones.* Sobre un muestreo de 100 soluciones iniciales con el GRASP, se calculó el valor medio de rigidez. Este proceso se repitió para instancias con n = 15, 20,... 120. Se graficó la rigidez promedio contra el número de vértices.

- *Tiempo de Ejecución.* Sobre las mismas 100 muestras se calculó el tiempo promedio para cada número de vértices y también se graficó contra el número de vértices, para ambos métodos.

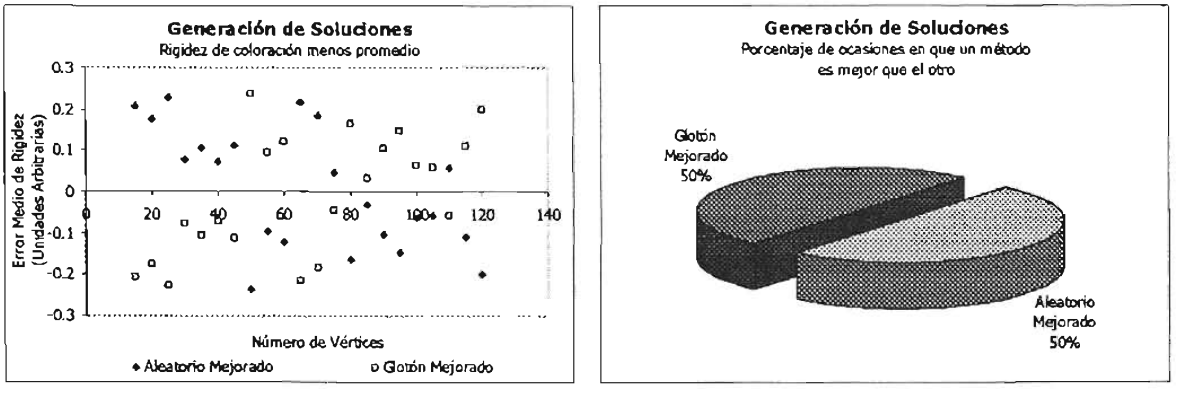
Resultados y Análisis. En todas las graficas los rombos representan la Solución Inicial Aleatoria Mejorada y los cuadrados representan la Solución Inicial Glotona Mejorada.

- *Calidad de las Soluciones.* Se muestra en la gráfica 4.1. el comportamiento promedio de la rigidez en la solución inicial contra el número de vértices. Podemos observar en ella que la calidad de las soluciones obtenidas con los algoritmos Aleatorio Mejorado y Glotón Mejorado es muy similar.



Gráfica 4.1. No es evidente alguna ventaja entre alguna de las técnicas.

Si tomamos los residuales de las dos soluciones, es decir, graficamos el valor de cada variable menos el promedio para cada instancia, podemos ver si hay alguna ventaja de usar alguna de las técnicas, por mínima que ésta sea, y es lo que se muestra en la *gráfica 4.2a.* Si tomamos el número de veces en que una técnica es mejor que otra obtenemos la grafica de pastel *4.2b.*

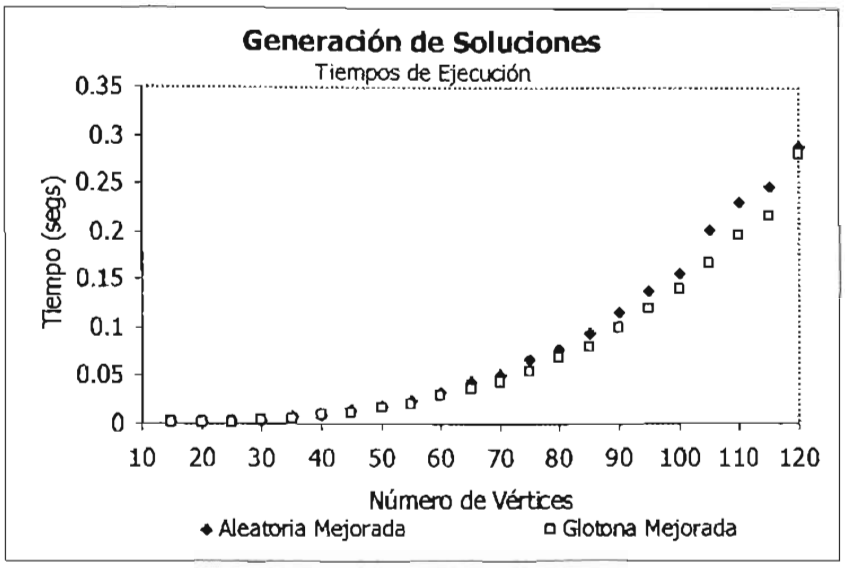


(a) *(b)*

Gráfica 4.2. Error promedio de la rigidez de coloración y porcentaje en que una rigidez promedio es mejor que la otra.

Es claro que en lo concerniente a la calidad de resultados, ambas técnicas son equivalentes.

- *Velocidad de los Algoritmos.* Como se puede ver de la gráfica, la técnica glotona mejorada es ligeramente más rápida que la aleatoria mejorada.



Gráfica 4.3. Tiempo de ejecución de cada instancia con respecto al número de vértices. El Generador Glotón Mejorado es ligeramente más rápido.

Recomendaciones.

- Aunque ambos métodos dan resultados muy similares, el Método Glotón Mejorado es ligeramente más rápido. Así, este es el recomendado para su uso en general.

4.2. Método de Mejora.

Búsqueda Local. Se intentaron muchas técnicas de mejora[1], pero la única que no generaba ninguna preferencia con respecto al número de colores utilizados o al orden en que se probaban los vértices, fue un método de Búsqueda Local, que selecciona cualquier vértice no mejorado y prueba en él todos los colores disponibles aleatoriamente[2].

Descripción del Procedimiento y sus Variables. Empezando por una coloración por depurar *Col(i)*, seleccionamos algún vértice no mejorado, y probamos todos los colores posibles al azar en ese vértice hasta tener una mejoría. Si se agotan los colores y no se mejora, se deja la solución como estaba y se prueba con otro vértice. El proceso termina cuando ya no se puede mejorar más la coloración al haber probado todos los colores en todos los vértices.

[1] Entre otras, mejoras de colores y de vértices secuenciales, permutación de posiciones de dos vértices, mutación de color al azar, entre otras.

[2] Si la prueba de colores no se realiza al azar, los grafos terminan pintados sesgadamente, es decir, se tienen mayor cantidad de colores 1 y 2 que de *n-1* y *n*, ya que se prueban colores hasta obtener la primera mejora. Esto nos llevan a soluciones que convergen más lentamente, a soluciones no tan buenas como la prueba de colores y vértices totalmente al azar.

Pseudo Código del Método de Búsqueda Local.

```
INICIO {Procedimiento}
    Leer una coloración por mejorar Col(j).
    MIENTRAS tengamos una mejora en la solución:
        Generamos una secuencia de visita de vértices seq(j)
        DESDE j=1 a n
            Generamos una secuencia de prueba de colores SeqCol(i)
            DESDE i=1 a k
                Si Col(seq(j))= SeqCol(i) reduce la rigidez sin aumentar la
                incompatibilidad, o reduce la incompatibilidad sin aumentar la
                rigidez reemplazar el color actual del vértice por SeqCol(i)
            FIN {Desde i}
        FIN {Desde j}
    FIN {Mientras}
FIN {Procedimiento}
```

Un comentario sobre complejidad computacional. Este elemento es el que mayor complejidad computacional tiene, la cual es del orden de $O(n^3)$; así, ésta es la complejidad computacional del GRASP. Para el algoritmo de Búsqueda Dispersa, debido a que se exploran del orden de n^2 soluciones, la complejidad final del algoritmo es de $O(n^5)$. Este, por supuesto, es el caso más pesimista, ya que en la práctica el comportamiento promedio de nuestro algoritmo BD es el de una función cúbica. En muchas ocasiones la complejidad computacional (que siempre es el peor caso), no es indicativo del comportamiento de un algoritmo en la realidad[3].

4.3. Método de actualización del RefSet.

Dos características por considerar dentro del RefSet. En esta sección más que comparar diferentes métodos se probaron diferentes parámetros dentro del mismo:

- ***Tamaño del RefSet.*** Se consideraron cuatro tamaños de conjunto de referencia: 5, 10, 15 y 20 elementos.

- ***Proporción de Calidad / Diversidad.*** Las soluciones de calidad son aquellas coloraciones válidas cuya rigidez es mínima; las soluciones de diversidad son aquellas que tienen uno o dos vértices no válidos y mínima rigidez. Se probaron 5 diferentes ponderaciones entre ambos conjuntos: 10/90, 30/70, 50/50, 70/30, 90/10.

Descripción del Procedimiento y sus Variables. Cada una de las coloraciones obtenidas en cierta generación, ya sea por solución inicial o por mezcla de soluciones con mejora (para

[3] En la Investigación de Operaciones es ampliamente conocido el caso del Método Simplex, que es la técnica fundamental de solución numérica para un problema de Programación Lineal. La complejidad computacional en el peor caso es exponencial, sin embargo para una instancia promedio este algoritmo es asombrosamente eficiente.

generaciones del *RefSet* subsecuentes), es incluida en el *BigSet* y es probada para ser incluida en el *RefSet* de dicha generación. *RefSet_Size* es el número de elementos en el *RefSet*, *Quality_Size* es el número de elementos de alta calidad por incluir al *RefSet*, *QSet* es un subconjunto con los mejores elementos de calidad y *DSet* es un subconjunto con los mejores elementos de diversidad. *QSet* y *DSet* incluye los 20 mejores candidatos para el nuevo *RefSet* en cada categoría.

Pseudo Código del Método de Actualización del RefSet.

INICIO {Procedimiento Principal}
Tamaño de *QSet* y *DSet* = 20 elementos
Tamaño de *RefSet* = *RefSet_Size* elementos
MIENTRAS haya elementos en el *BigSet*
Llamar al procedimiento *Lista_de_Mejores*
FIN {Mientras}
Para todos los elementos del *RefSet*
Llamar al procedimiento *Lista_de_Mejores*
FIN {Para todos}
Vaciar el *RefSet*
Tomar los *Quality_Size* mejores elementos de *QSet* e incluirlos en el *RefSet*
Tomar los (*RefSet_Size* - *Quality_Size*) elementos de *DSet* e incluirlos en el *RefSet*
Vaciar *BigSet*
FIN {Procedimiento Principal}

INICIO {Procedimiento *Lista_de_Mejores*}
Si tiene todos los vértices bien coloreados y una rigidez menor que el peor elemento de *QSet*, y no hay un elemento con una rigidez exactamente igual en valor, incluirlo y ordenarlo (de menor a mayor), si no, terminar el procedimiento.
Si tiene uno o dos vértices mal coloreados y una rigidez menor que el peor elemento de *DSet*, y no hay un elemento con una rigidez exactamente igual en valor, incluirlo y ordenarlo (de menor a mayor) en el, si no, terminar el procedimiento.
FIN {Procedimiento *Lista_de_Mejores*}

Un comentario sobre coloraciones equivalentes. Como ya se mencionó, en el problema de coloración mínima y robusta existen coloraciones equivalentes, citemos el ejemplo 1.3 de Programación de Exámenes, una coloración válida del grafo es: *(1, 2, 3, 2, 1, 3)*, la cual es equivalente a *(3, 2, 1, 2, 3, 1)*, porque las entradas donde hay "1's" (primera y quinta) fueron substituidas por "3's" y en donde habia "3's" (tercera y sexta entradas del vector) ahora hay "1's". Otra coloración equivalente es *(1, 3, 2, 3, 1, 2)*, que se obtiene intercambiando de lugar los "2's" y los "3's" en la primera coloración.

Hay otras tres coloraciones equivalentes: *(3,1,2,1,3,2)*, *(2,1,3,1,2,3)* y *(2,3,1,3,2,1)*. De hecho, dada una grafica pintada con *k* colores se tienen *k!* coloraciones equivalentes[4]. Todas estas coloraciones equivalentes tienen los mismos vértices pintados con los mismos colores y tienen exactamente la misma rigidez. Se desarrolló un vector de equivalencias el cual en la fila superior se colocan los colores ordenados secuencialmente *(1,2,3,...k)* y en la segunda fila se coloca el color equivalente en el segundo vector de coloración, si dicha relación es uno a uno, las coloraciones son equivalentes. Mostramos a continuación las coloraciones equivalentes de algunos de los vectores de coloración antes mencionados:

	(1,2,3,2,1,3)		
	1	**2**	**3**
(1,2,3,2,1,3)	*1*	*2*	*3*
(3,2,1,2,3,1)	*3*	*2*	*1*
(1,3,2,3,1,2)	*1*	*3*	*2*

(3,1,2,1,3,2)		
1	**2**	**3**
2	*3*	*1*
2	*1*	*3*
3	*2*	*1*

(2,3,1,3,2,1)		
1	**2**	**3**
3	*1*	*2*
1	*3*	*2*
2	*1*	*3*

Como contraste, los vectores de coloración *(1,2,3,2,1,3)* y *(1,2,3,1,2,3)* NO son equivalentes, ya que en el primer vector la primera y la quinta entrada se pintan del mismo color, lo mismo que la segunda y la cuarta, cosa que no ocurre en el segundo vector de coloración.

Un método mas sencillo para detectar coloraciones equivalentes sin cálculos extra, es simplemente comparar la rigidez y el número de aristas incompatibles de ambos vectores, si ambos números son iguales, las coloraciones son equivalentes o muy parecidas, como se puede ver en el ejemplo paso a paso del próximo capítulo.

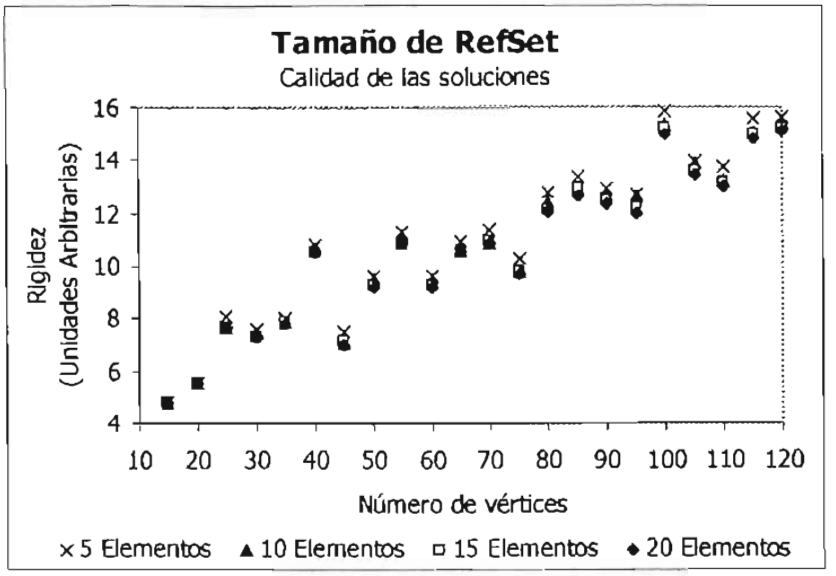
Por otra parte, la rigidez también nos sirve como un eficiente vector *hash(x)*, vector que, como se mencionó en el capitulo anterior, nos permite calcular la "distancia" entre coloraciones, (es decir, a medida que la rigidez de dos soluciones se alejan en valor, sus coloraciones se parecen menos entre si). De hecho, en la práctica, las soluciones de calidad y de diversidad se alejan a medida de que el *RefSet* madura: por tener un par de aristas incompatibles, los elementos de diversidad tienden a valores de rigidez menores que los elementos de calidad, que solo contiene coloraciones válidas. Dicho de otra forma, las soluciones de calidad tienen menos "grados de libertad", y su rigidez es mayor.

Pruebas Comparativas para el Método de Actualización del RefSet. Se corrió cada algoritmo 10 veces por cada instancia y las características por evaluar fueron la rigidez promedio y tiempo medio de ejecución, ambas características se graficaron contra número de vértices.

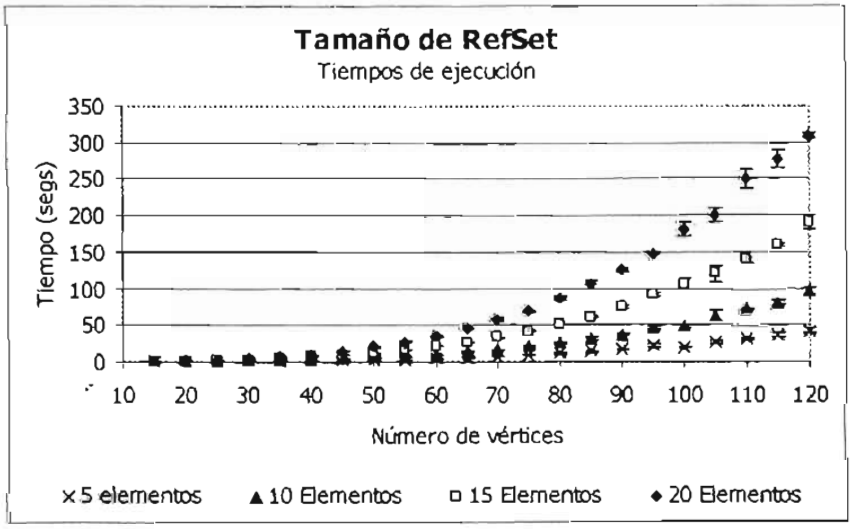
[4] Ordenemos los k colores cromáticamente (rojo el primero, y violeta el ultimo, por ejemplo). Al primer color de la lista le podemos asignar *k* números, al segundo *k-1*, al tercero *k-2* etc. Así, podemos tener *k!* coloraciones equivalentes.

Resultados y Análisis.

- ***Tamaño del RefSet.*** Se muestra en la *gráfica 4.4.* la rigidez con respecto al número de vértices. En la gráfica 4.5. se muestra el tiempo de ejecución para los cuatro tamaños de *RefSet* (5, 10, 15 y 20 elementos).



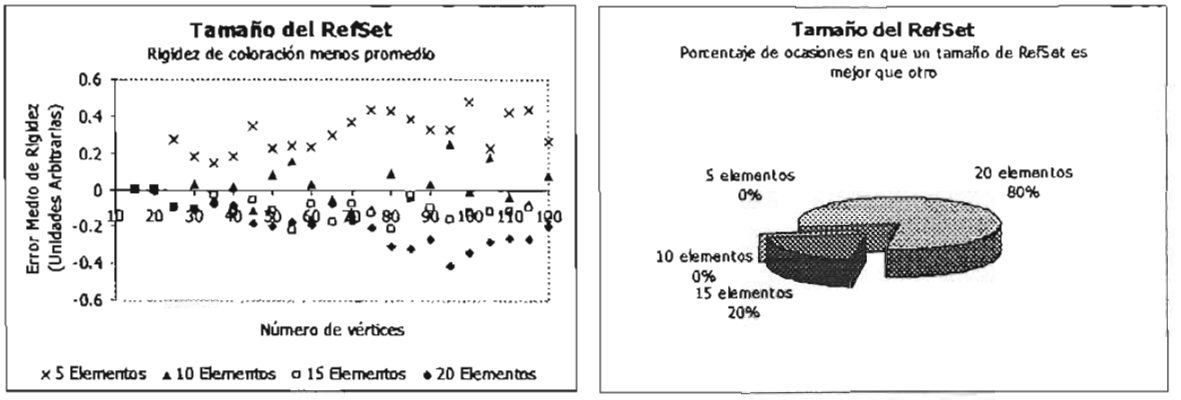
Gráfica 4.4. Rigidez media obtenida para cada tamaño de instancia de acuerdo al tamaño del Conjunto de Referencia.



Gráfica 4.5. Tiempo de ejecución medio requerido para cada tamaño de Conjunto de Referencia

En la gráfica 4.4 solo se puede apreciar una muy pequeña desventaja para el *RefSet* de cinco elementos. Para evidenciar cualquier diferencia, se muestra en la gráfica 4.6a el error medio de rigidez, donde podemos apreciar claramente que un tamaño de cinco

elementos nos da la peor rigidez para cualquier tamaño de instancia. Incluso parece haber una desventaja con un conjunto de referencia de diez elementos. Para eliminar cualquier duda, veamos de todas las ocasiones que se corrió el algoritmo cuantas veces un tamaño de *RefSet* es mejor que otro, dichos resultados se muestran el la *gráfica 4.6b.*

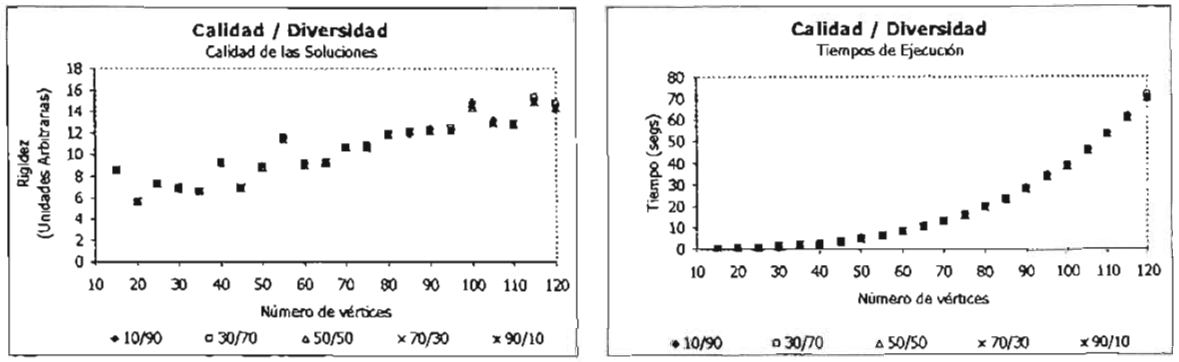


(a) *(b)*

Gráfica 4.6. *Error promedio de la rigidez de coloración y porcentaje en que una rigidez promedio es mejor que la otra, respecto al tamaño de Conjunto de Referencia*

En la gráfica de porcentajes era de esperarse que un mayor conjunto de referencia tenga una ventaja (simplemente porque un *RefSet* más grande explora más elementos), lo que es de llamar la atención es que un *RefSet* más grande no siempre es la mejor opción. Si observamos de nuevo la *gráfica 4.6a*, vemos que da lo mismo usar un *RefSet* de 15 o de 20 elementos, para tamaños de instancia menores a 70 vértices. Dicho de otra forma, no se aprecia ninguna ventaja entre tomar 15 o 20 elementos de *RefSet* cuando las instancias son menores a 70 vértices.

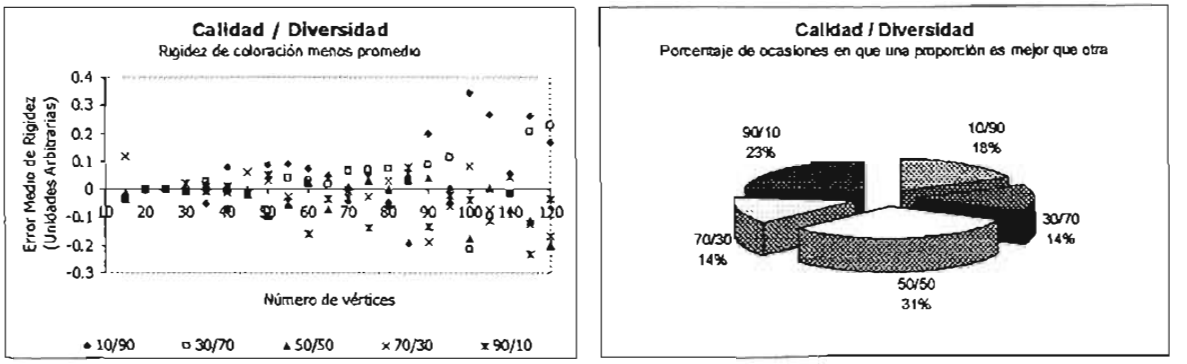
- ***Proporción entre Calidad / Diversidad.*** Ahora se muestran las graficas de frecuencia con respecto a los porcentajes entre calidad y la diversidad, se tomaron 5 proporciones: 10/90, 30/70, 50/50, 70/30 y 90/10.



(a) *(b)*

Gráfica 4.7. *Rigidez de las soluciones y tiempo de ejecución respecto a la proporción entre calidad y diversidad.*

No se puede apreciar ninguna tendencia de mayor calidad para alguna proporción en la *gráfica 4.7a*, y tampoco se aprecia ninguna diferencia en tiempos de compilación (*figura 4.7b*). En la *gráfica 4.8a* tampoco se puede ver ninguna tendencia de las soluciones. Sin embargo, en la gráfica de porcentajes *4.8b* sí hay una ligera tendencia a favor de la proporción 50/50.



(a) *(b)*

Gráfica 4.8. *Error promedio de la rigidez de coloración y porcentaje en que la rigidez promedio de una proporción es mejor que las demás, respecto al tamaño de Conjunto de Referencia*

Recomendaciones.

- *En grafos de menos de 70 vértices:* 15 elementos en el *RefSet* y una proporción de 50/50 entre calidad y diversidad son las mejores opciones.

- *En grafos de más de 70 vértices:* Se recomienda 20 elementos en el *RefSet* y una proporción de 50/50 entre calidad y diversidad.

4.4. Método de Generación de Subconjuntos.

Búsqueda exhaustiva en el subconjunto de dos elementos. En [Laguna, 2004] se recomienda dedicar la mayor cantidad del poder de cálculo a la combinación de subconjuntos de dos elementos. En base a este criterio y a la experiencia empírica que se tuvo en el desarrollo de este algoritmo, se decidió realizar una búsqueda exhaustiva sobre todos los subconjuntos posibles de dos elementos. Si tenemos *b* elementos en el *RefSet*, el número de pares posibles es *b(b-1)/2*. Este subconjunto de dos elementos es investigado exhaustivamente. Por otro lado, se decidió tomar también los cuatro mejores elementos del *RefSet* y hacer sus subconjuntos de tres elementos, porque si tomáramos todas las posibles combinaciones de subconjuntos de tres tendríamos *b(b-1)(b-2)/6*, las cuales son

muchísimas más soluciones que las de los subconjuntos de dos elementos. De esta forma se explota plenamente el conjunto de dos elementos, sin dejar de lado el de tres.

4.5. Método de Combinación de Soluciones.

Un método para hacer religado y en cuantos pasos. En esta sección es donde el Religado de Trayectorias cobra la mayor importancia, ya que surgen las siguientes preguntas: cuando mezclamos dos o más soluciones, ¿bajo qué criterio decidimos en qué secuencia incluiremos los elementos de una solución dentro de otra? Y no solo eso: ¿qué tanto porcentaje de una solución se pondrá en otra y en cuantos "trozos" se mezclaran?, dicho de otra forma, ¿en cuántos "pasos" llegaremos de una solución a otra?

- ***¿Cuántos pasos entre soluciones?*** En este trabajo se estudió el comportamiento para 5, 10, 15 y 20 "pasos" para llegar de la solución inicial a la final.

- ***Matriz de Adyacencia Cúbica contra Inclusión Aleatoria.*** Dos métodos fueron utilizados para realizar religado de trayectorias entre soluciones: uno considera la matriz de adyacencia de aristas elevada al cubo (la cual nos muestra de cuántas formas podemos llegar de un vértice a otro usando tres aristas). Eligiendo al azar cualquier vértice de esta matriz cúbica, el vértice con el valor más alto se visitara primero, el siguiente en valor seguirá y así sucesivamente. El otro método, el de Inclusión Aleatoria, nos dice que una vez decidido el número de "pasos" por realizar entre la solución inicial y la final, en el primero, a cada vértice del grafo inicial le asignamos una probabilidad de *1/s* (donde *s* es el número de pasos) de incluir el valor del vértice final en ese paso, en el siguiente se da una probabilidad de *2/s* de incluirlo y así sucesivamente.

  Utilizando nuevamente el ejemplo de la sección 1.3. se hará el religado entre dos distintas coloraciones, por ejemplo, los vectores de coloración (1,2,3,2,1,3) y (2,3,1,1,3,2), las cuales son coloraciones válidas y tienen distinta rigidez. Se hará un ejemplo de religado para cada técnica.

  Religado con matriz de Adyacencia Cúbica. La matriz de adyacencia asociada al ejemplo de la sección 1.3. y su correspondiente valor al cubo se muestran a continuación:

$$a = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array} \qquad a^3 = \begin{bmatrix} 2 & 6 & 6 & 7 & 3 & 3 \\ 6 & 2 & 6 & 3 & 3 & 7 \\ 6 & 6 & 2 & 3 & 7 & 3 \\ 7 & 3 & 3 & 2 & 6 & 6 \\ 3 & 3 & 7 & 6 & 2 & 6 \\ 3 & 7 & 3 & 6 & 6 & 2 \end{bmatrix}$$

En la matriz cúbica vemos qué tan relacionado están los vértices entre sí. Por ejemplo, el vértice *a* está fuertemente relacionado con el *d* (7 unidades) y un poco menos con el *c* y el *b* (6 unidades) y su influencia es bastante menor con los vértices *e* y *f* (3 unidades).

Asociada a la matriz a^3, podemos generar la matriz de secuencia por importancia *Sec* en donde ordenamos por filas a los vértices de los más relacionados a los menos. Para la matriz a^3 del ejemplo, su matriz de secuencia es:

$$Sec = \begin{bmatrix} a & d & b & c & e & f \\ b & f & a & c & d & e \\ c & e & a & b & d & f \\ d & a & e & f & b & c \\ e & c & d & f & a & b \\ f & b & d & e & a & c \end{bmatrix}$$

Si empezamos a mezclar las soluciones por un vértice cualquiera, por ejemplo por el e, iniciamos con *e* y de ahí seguimos con el *c*, el *d*, el *f*, el *a* y finalmente el *b*. La mezcla de los vectores de coloración anteriores empezando arbitrariamente por la fila *c*, su religado en dos pasos sería:

(1,2,3,2,1,3)	*Solución Inicial*
(1,2,**1**,2,**3**,3)	*Paso 1*
(**2**,**3**,**1**,2,**3**,3)	*Paso 2*
(**2**,**3**,**1**,**1**,**3**,**2**)	*Solución Final*

Obviamente, a ambos pasos se les aplica el método de mejora.

Religado por Inclusión Aleatoria. En esta forma la mezcla de soluciones es mucho mas sencilla, ya que solo hay que definir una secuencia de inclusión, como por ejemplo (*f,c,a,b,e,d*). Si hacemos la inclusión en 5 pasos tendríamos:

(2,3,1,1,3,2)	*Solución Inicial*
(2,3,1,1,3,**3**)	*Paso 1*
(2,3,**3**,1,3,**3**)	*Paso 2*
(**1**,3,**3**,1,3,**3**)	*Paso 3*
(**1**,**2**,**3**,1,3,**3**)	*Paso 4*
(**1**,**2**,**3**,1,**1**,**3**)	*Paso 5*
(**1**,**2**,**3**,**2**,**1**,**3**)	*Solución Final*

De nuevo, a la coloración obtenida en cada paso se le aplica el método de mejora. Si el (*número de pasos+1*) y el número de soluciones no es un número entero como en los ejemplos, simplemente se hace un redondeo y se toma esa cantidad de elementos por paso.

Descripción del Procedimiento y sus Variables. En ambos casos seleccionamos cierta secuencia de inclusión, ya sea por la matriz de *Sec*, o azarosamente. El religado de la trayectoria que va de la solución inicial a la final, se hará en cierta cantidad de pasos (variable *partitions*).

1. Pseudo Código de la Mezcla de Soluciones por Matriz de Adyacencia Cúbica.

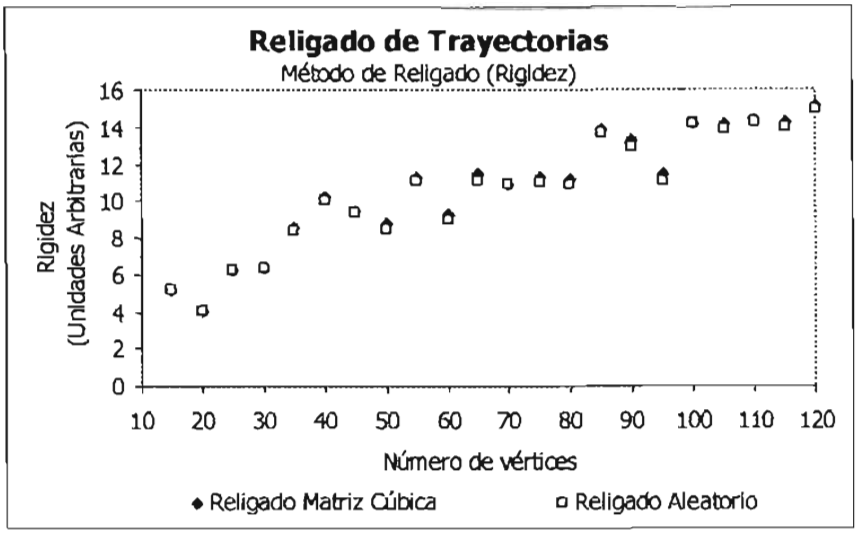
```
INICIO {Procedimiento}
    Calcular la matriz de adyacencia cúbica (Or3) y la matriz de orden (Sec)
    DESDE a=1 hasta n-1
        DESDE b=a+1 hasta n
            Seleccionar cualquier fila de la matriz Sec al azar
            DESDE m=1 a partitions
                Cambiar el valor de (n*m/partitions) elementos en Col_a de acuerdo al
                    orden de la fila seleccionada en Seq con elementos de Col_b
                Mejorar la solución
            FIN {Desde m}
        FIN {Desde b}
    FIN {Desde a}
FIN {Procedimiento}
```

2. Pseudo Código de la Mezcla de Soluciones por Inclusión Aleatoria.

```
INICIO {Procedimiento}
    Desde a=1 a n-1
        Desde b=a+1 a n
            Generamos una secuencia aleatoria de mezcla Sec
            Desde m=1 a partitions-1
                    Orden=Valor redondeado de (m*n/partitions)
                    Desde j=1 a Orden
                        Col_a(Sec(j)) = Col_b(Sec(j))
                    Fin {Desde j}
                    Mejorar la nueva solución Col_a
            End {Desde m}
        Fin {Desde b}
    Fin {Desde a}
Fin {Procedimiento}
```

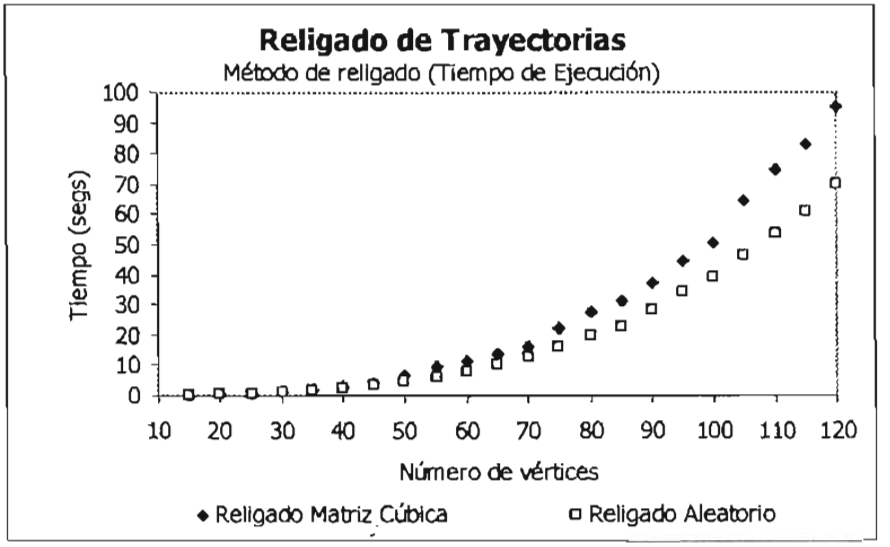
Resultados y Análisis.

- ***Matriz Cúbica v.s. Inclusión Aleatoria.*** Se comparó la rigidez obtenida respecto al número de vértices con ambos procedimientos, como se muestra a continuación:



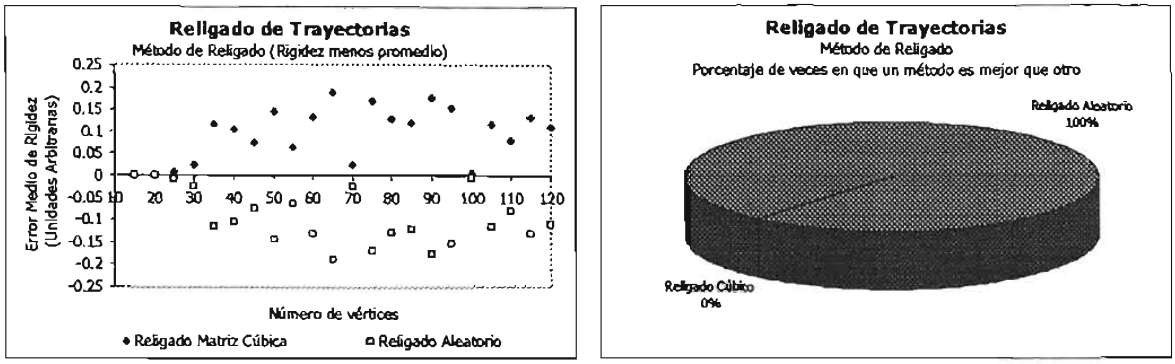
Gráfica 4.9. Rigidez respecto al método de religado utilizado.

En la *gráfica 4.9.*, aparentemente el Religado Aleatorio es ligeramente mejor que el Cúbico, por otro lado, En la *gráfica 4.10.* se puede apreciar que también el religado aleatorio es una técnica mas rápida que la de Matriz Cúbica.



Gráfica 4.10. Tiempo de ejecución respecto al método de religado utilizado.

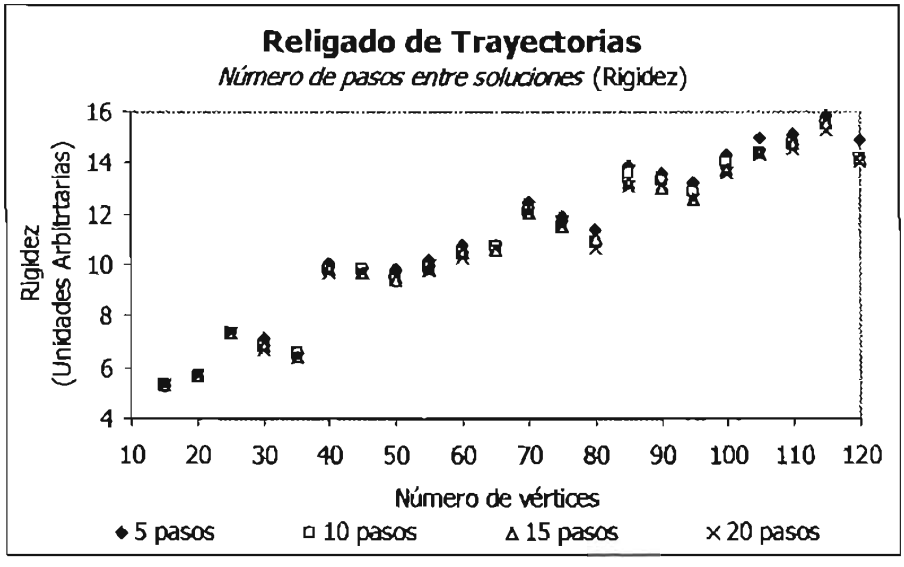
Finalmente, tanto en las *gráficas 4.11 a* y *b* se puede ver que en lo que respecta a calidad de soluciones, el religado aleatorio es mejor, debido a que en todas y cada una de las ocasiones su desempeño fue superior respecto al religado de matriz cúbica.



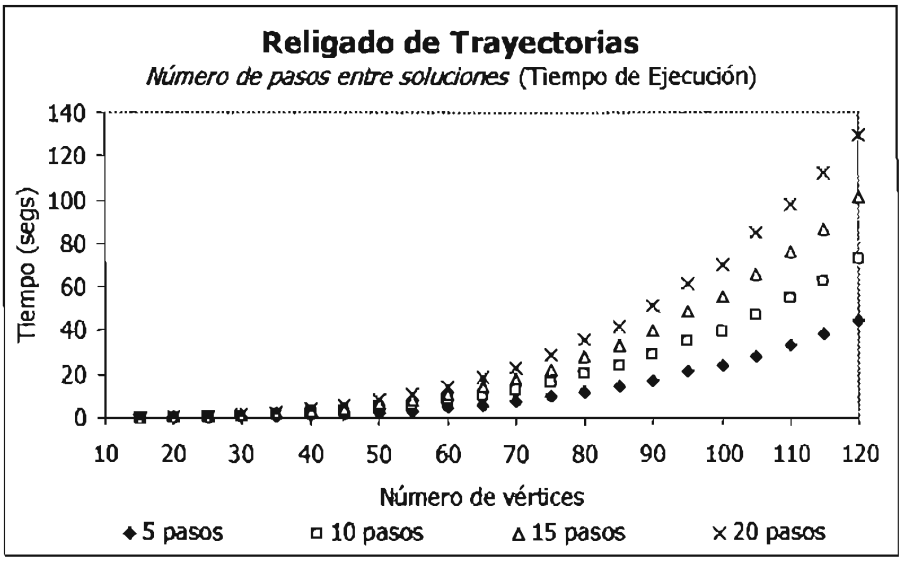
(a) *(b)*

Gráfica 4.11. *Error promedio de la rigidez de coloración y porcentaje en que la rigidez promedio de un método es mejor que el otro.*

- ***Número de pasos.*** Como se dijo anteriormente, intentamos cuatro cantidades diferentes: 5, 10, 15 y 20 pasos. En la *gráfica 4.12.* se muestra la rigidez promedio contra número de vértices:

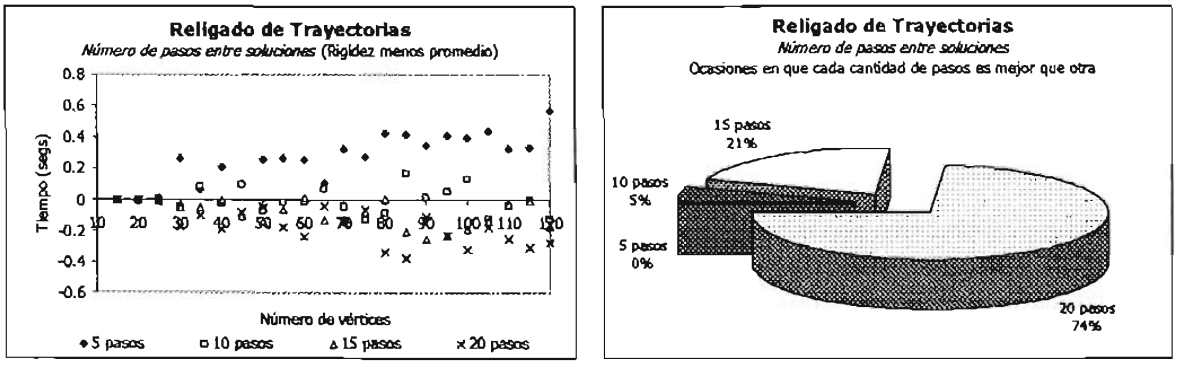


Gráfica 4.12. Rigidez promedio respecto al número de pasos.



Gráfica 4.13. Tiempo de ejecución respecto al número de pasos.

En la *gráfica 4.14a.* se puede ver que utilizar 5 pasos o incluso 10 no tiene tan buen desempeño como usar 15 o 20 pasos; el mejor desempeño se obtiene con estos dos últimos. En la *gráfica 4.14b* se hace evidente que 20 pasos es mucha mejor opción.



(a) *(b)*

Gráfica 4.14. *Error promedio de la rigidez de coloración y porcentaje en que la rigidez promedio para cuatro diferentes cantidades de pasos.*

Recomendaciones. Hemos comparado los procedimientos de religado de trayectorias usando la matriz de adyacencia cúbica y haciendo una inclusión aleatoria, también consideramos 5, 10, 15 y 20 pasos para hacer dicho religado, a continuación mostramos las recomendaciones:

- El religado se debe realizar con una Matriz Aleatoria y 20 pasos entre soluciones.

Pseudo Código preliminar para el Algoritmo de Búsqueda Dispersa en el Problema de Coloración Robusta

Pseudo código preliminar de Búsqueda Dispersa para Coloración Robusta.

{Paso 1: GRASP para Generación de Soluciones.}
DESDE *i=1* a *100*
    Generar una Solución Glotona Mejorada (Ver sección 4.1)
    Usar en ella el Método de Búsqueda Local (Sección 4.2.)
    Incluir todas las soluciones mejoradas en el *BigSet*.
FIN {Desde}

{Paso 2: Método de actualización del RefSet}
El conjunto de referencia deberá tener 20 elementos.
Eliminar del *BigSet* todas las soluciones redundantes.
Seleccionar 50% de soluciones con coloración válida y mínima rigidez.
Seleccionar el otro 50% de las soluciones con uno o dos vértices mal coloreados y mínima rigidez.
Vaciar *BigSet*.

{Paso 3: Método de Generación de Subconjuntos}
Considerar todas las combinaciones posibles de conjuntos de dos elementos.
Considerar la mezcla de los mejores 4 elementos en los conjuntos de 3 elementos.

{Paso 4: Método de Combinación de Soluciones}
Para todos los subconjuntos de dos elementos a y b
    Desde *m=1* hasta *Partitions*
        Mezclar aleatoriamente los elementos de la solución *col(b)* en *col(a)* (ver Sección 4.5)
        Usar el método de mejora en la nueva solución
        Agregar la solución al *BigSet*
    Fin {Desde}
Fin {Para todos}
Para los cuatro mejores elementos del *RefSet*
    Mezclar los tres elementos en una proporción de 1/3 cada uno
    Usar en las soluciones el método de mejora y agregarlas al *BigSet*
Fin {Para cuatro}
Actualizar el *RefSet* con las mejores soluciones del *BigSet*
Si por lo menos hemos encontrado una mejor solución en el *RefSet* de esta generación, regresar al paso 2, si no, terminar el algoritmo.

Comentarios sobre el Algoritmo.

En el primer paso, *Método de Generación de Soluciones*, la mejor opción es el *procedimiento Glotón Mejorado* porque es ligeramente más rápido que el aleatorio

mejorado y de resultados equivalentes. Para todos los tamaños de instancias, el *Método de Búsqueda Local* para hacer mejoras es rápido y no genera preferencias respecto a algún vértice o color.

En el *Método de Actualización del RefSet*, El algoritmo es poco sensible a la proporción entre calidad y diversidad, pero se comporta un poco mejor para la proporción 50/50. Por otro lado un conjunto de referencia de 20 elementos da las mejores soluciones.

En el *Método de Generación de Subconjuntos* se consideraron todos los subconjuntos de dos elementos y las combinaciones de los cuatro mejores elementos en los conjuntos de tres elementos. Finalmente, en el *Método de Combinación de Soluciones* el religado de trayectorias Aleatorio es el elegido, por calidad y rapidez.

En este capítulo hemos optimizado los elementos constitutivos de los algoritmos, el siguiente paso es hacerlos funcionar armoniosamente para obtener aún mejores soluciones, el cual es el objetivo del siguiente capítulo.

Capítulo 5. Implementación de los Algoritmos.

Un Algoritmo es mucho más que la suma de sus partes. Así, al encontrar los elementos idóneos de ambos algoritmos, hemos dado solo el primer paso: necesitamos hacer que trabajen en conjunto. En el capítulo anterior se optimizaron los "bloques constructores" del GRASP y la Búsqueda Dispersa. Estos elementos se deben integrar, y mejorar su desempeño dentro de cada algoritmo como un todo.

A continuación se describe la "mejora continua" realizada en los algoritmos GRASP y de Búsqueda Dispersa del capítulo anterior, para que obtengan mejores resultados en tiempos de ejecución equivalentes. Se presentan los pseudo códigos asociados a las mejoras.

Las soluciones de un algoritmo meta-heurístico iterativo se van acercando a un cierto valor a cada generación que pasa. Este proceso es llamado convergencia de soluciones. De acuerdo a [Glover [b], 2004], la técnica de Búsqueda Dispersa converge exponencialmente (es decir, una exponencial decreciente es la función que más se le aproxima a la serie de datos). Se presentan ejemplos donde se corrobora esta aseveración para diferentes tamaños de instancia: 30, 60, 90, 120 vértices.

Con el propósito de aclarar conceptos se presenta un ejemplo numérico para una instancia con 9 vértices, en la primera parte se usa el GRASP para generar una solución inicial y posteriormente con base en un par de soluciones desarrollamos un ciclo de Búsqueda Dispersa.

En este capítulo se hace una revisión a las implementaciones finales de los algoritmos desarrollados. En la sección 5.1 se muestran los puntos de mejora y el beneficio obtenido con respecto al algoritmo original, y un diagrama de la implementación final. En la sección 5.2 mostramos los pseudo-códigos finales de los algoritmos. En la sección 5.3 se desarrolla un ejemplo sencillo paso a paso.

5.1. Puntos de Mejora

Punto de Mejora 1: Un Glotón más Eficaz. En el capítulo anterior generamos un buen GRASP, con un glotón que encuentra una coloración válida, y la mejora. Tomemos como ejemplo, su desempeño en una instancia de 100 vértices: el glotón genera una coloración inicial válida no robustecida en tan solo 0.01 segundos con una rigidez promedio de *110.4±6.4* unidades. Esta coloración es mejorada, mediante muchos esfuerzos, a un valor de rigidez de *16.9±1.3* unidades, después de 0.59 segundos de trabajo en promedio. Estos datos, en principio, hablan muy bien del algoritmo, ya que entrega una buena solución en menos de un segundo de una instancia que ya es considerada "grande". A partir de los tiempos de ejecución, nos damos cuenta que el 1.5% del tiempo de ejecución se gasta en obtener el glotón, y el 98.5% restante se busca mejorar dicha solución. ¿No sería mejor repartir el trabajo más equitativamente, buscando que el glotón entregue una mejor solución y con esto, el método de mejora tenga que cargar menos?

El glotón utilizado es un glotón idéntico a los usados en el problema de coloración mínima, sin considerar la rigidez. El objetivo será ahora, encontrar un glotón que además de encontrar una coloración válida o con pocos arcos conflictivos, tome en consideración la búsqueda de una coloración parcial con menor rigidez.

Se propone un glotón que en la coloración parcial, tome un color cuya rigidez asociada con ese vértice, sea mínima. De esta forma, solo consideraremos los colores que a cada paso hagan válida la coloración y que tengan una rigidez menor.

Pseudo Código de la Solución Inicial Glotona 2

```
INICIO {Procedimiento}
    DESDE m=1 a BigSetSize
        Generar una secuencia en que los vértices serán "visitados": Sec(j)
        DESDE j=1 a n
            DESDE i=1 a k
                Si Col(Sec(j)) = i es una coloración válida, poner esta solución en una
                    lista provisional
            FIN {Desde i}
            Tomar la solución de la lista provisional con menor rigidez.
            Si la lista provisional esta vacía, pintar el vértice con un color al azar.
        FIN {Desde j}
    FIN {Procedimiento}
```

Tomando la misma instancia, la mejora se hace evidente: el algoritmo obtiene ahora una solución, en promedio, en 0.15 segundos, la coloración del glotón tiene una rigidez promedio de 24.6±1.8 unidades y se mejora hasta 16.9±1.2, lo cual es equivalente al valor anterior promedio de rigidez, en tan solo una cuarta parte del tiempo requerido. Así hemos hecho un reparto de las cargas de trabajo de una forma más equitativa: el glotón ahora consume el 15% del tiempo y la mejora consume el 85% del tiempo.

Otra forma de repartir el trabajo no es solo cargarle más al que menos hace, sino también disminuir trabajo innecesario o poco productivo del método de mejora, que fue lo que se realizó en el siguiente punto.

Punto de Mejora 2: Búsqueda Local más rápida. Dentro del mundo de la programación es muy famoso un viejo artículo de Edsger Dijkstra[1] donde categóricamente dice: "la calidad de un programador decrece en función del número de veces que utiliza la función 'goto' en los programas que produce". Para todos los programadores que nos sentimos orgullosos de lo bien "estructurados" que son nuestros programas, el uso del *"goto"* esta prohibido. Pero hay casos, como el de esta mejora, en donde el *goto* no es la única solución, pero es, la "ruta más corta" para hacer la mejora.

Si observamos con atención el algoritmo de mejora del capítulo anterior podemos observar que al hacer la búsqueda de colores sobre un mismo vértice, con tal de no romper el *loop*, se prueban todos los colores antes de pasar al siguiente vértice. Este criterio es malo ya que podemos encontrar mas de un color que mejore la coloración, y solo uno aparecerá al final. Lo deseable es pasar al siguiente vértice al lograr una mejora, y con esto, reducir el tiempo de ejecución.

La solución "correcta" es incluir un *while* en el ciclo de revisión de colores: "Mientras no tengamos una mejora o se nos acaben los colores, continuar probando colores", pero haciendo esto, el algoritmo se ejecuta en más tiempo. La solución más rápida y cómoda, es incluir un "goto", es decir, hacer un "salto" que rompa el bucle, tal y como se muestra en el pseudo-código de esta sección.

Podemos ver para una instancia de 100 vértices que el tiempo de ejecución se reduce a la mitad, con solo incluir el *goto*: el tiempo promedio de ejecución del algoritmo anterior es de 0.15 segundos con rigidez de 16.9±1.2. En el mejorado se ejecuta en 0.07 segundos con una rigidez idéntica de 16.9±1.2

Pseudo Código de Búsqueda Local 2.

```
INICIO {Procedimiento}
    Leer una coloración por mejorar Col(j)
    MIENTRAS tengamos una mejora en la solución:
        Generamos una secuencia de visita de vértices seq(j)
        DESDE j=1 a n
            Generamos una secuencia de prueba de colores SeqCol(i)
                DESDE i=1 a k
                    Si Col(seq(j)) = SeqCol(i) reduce la rigidez sin aumentar la
                      incompatibilidad, o reduce la incompatibilidad sin aumentar la
```

[1] Edsger W. Dijkstra. *Go To Statement Considered Harmful*. Communications of the ACM. Vol. 11 No. 3. March 1968. pags. 147-148. Dijkstra también es famoso en el mundo de la optimización por su algoritmo de ruta mas corta.

```
                    rigidez reemplazar el color actual del vértice por SeqCol(i). Ir a
                    SALTO
                FIN {Desde i}
            SALTO:
            FIN {Desde j}
        FIN {Mientras}
FIN {Procedimiento}
```

Con las dos mejoras anteriores hemos reducido el tiempo de ejecución del GRASP en casi un 90 %, produciendo resultados equivalentes. Parece evidente que las mejoras van por buen camino. Ahora debemos buscar mejorar la calidad de las mismas.

Punto de Mejora 3: Templando la Solución sin Recocerla. Cuando se aplica el método de mejora, debemos tener cuidado en no dar demasiado peso a disminuir la rigidez: nos podemos encontrar con coloraciones con una rigidez minúscula, pero con muchas aristas incompatibles. Al encontrar el valor mínimo de peso a partir del cual las coloraciones son predominantemente válidas, uno no vuelve a preocuparse del problema y de hecho en el capítulo anterior ni siquiera se menciona esta ponderación entre validez y rigidez. En la función del capítulo anterior se utilizó $\kappa=1$:

$$f(c) = I(c) + \kappa \; R(c)$$

Pero ¿que pasaría si a partir de nuestra solución glotona, la "calentamos" un poco, por ejemplo seleccionando $\kappa=3$, y buscando rigideces mas pequeñas, a riesgo de generar algunas aristas incompatibles, y luego volvemos a "enfriar" la solución con $\kappa=1$ nuevamente, haciéndola válida aunque aumente su rigidez? La idea sonaba interesante, y se tomó la decisión de darle una oportunidad.

Del valor 16.9±1.2, valor que se obtenía en tan solo 0.07 segundos, cuando "templamos sin recocer" llegamos a un valor de 14.6±1.0 en 0.11 segundos en promedio. El costo es bajo contra el beneficio que se obtiene.

Punto de Mejora 4: ¿Cuántas veces se "templa" la solución? Una vez que se ha generado una solución glotona o se han mezclado dos soluciones, al aplicar el método de mejora, ¿cuántas veces conviene "templar" la solución para hacer un equilibrio entre eficiencia contra tiempo de ejecución? A partir de ensayo y error se pierde la eficiencia después de 3 o 4 ciclos de calentado y enfriado. Se decidió hacer cuatro ciclos de calentado y enfriado durante la fase de mejora del GRASP y por rapidez, tres ciclos de temple en la fase de mejora del religado de trayectorias.

Mejoras 1 a 4: beneficios al GRASP. A continuación se muestra una tabla donde se incluyen instancias *al(n)* muy grandes, entre 100 y 1000 vértices, y un estudio comparativo entre la rigidez de una coloración sin robustecer, y las robustecidas por tres métodos: Glotón sencillo, GRASP sin depurar (GRASP 1), GRASP perfeccionado (GRASP 2) se incluyen los tiempos promedio de ejecución en segundos.

Tabla 5.1. Comparativo de diferentes algoritmos sobre instancias muy grandes.

Instancia			*Rigidez en coloración válida (Rigidez / tiempo en segundos)*			
$G_{n,0.5}$	*n*	*k*	*Solución sin robustecer*	*Glotón*	*GRASP 1*	*GRASP 2*
al(100)	*100*	*34*	*111 ± 7 / 0.01*	*24.4 ± 1.7 / 0.02*	*16.8 ± 1.4 / 0.10*	***14.5 ± 0.9 / 0.12***
al(200)	*200*	*67*	*267 ± 11 / 0.11*	*41.6 ± 2.0 / 0.17*	*24.8 ± 1.5 / 0.95*	***22.9 ± 1.3 / 0.96***
al(300)	*300*	*100*	*427 ± 12 / 0.34*	*60.1 ± 1.8 / 0.58*	*32.3 ± 2.4 / 4.2*	***28.5 ± 1.2 / 3.6***
al(400)	*400*	*133*	*606 ± 15 / 0.82*	*82.3 ± 2.3 / 1.1*	*36.8 ± 1.8 / 10.5*	***35.2 ± 1.3 / 8.4***
al(500)	*500*	*167*	*799 ± 16 / 1.6*	*105.1 ± 2.6 / 2.6*	*49.1 ± 1.5 / 22.2*	***37.5 ± 1.8 / 16.1***
al(600)	*600*	*200*	*993 ± 17 / 2.7*	*128.7 ± 2.6 / 4.6*	*45.1 ± 1.6 / 38.3*	***43.1 ± 1.4 / 28.2***
al(700)	*700*	*233*	*1203 ± 22 / 4.3*	*152.5 ± 2.6 / 7.2*	*49.9 ± 2.1 / 62.7*	***47.7 ± 1.7 / 45.1***
al(800)	*800*	*267*	*1416 ± 24 / 6.5*	*175.4 ± 3.0 / 10.7*	*52.3 ± 1.9 / 96.1*	***50.8 ± 1.6 / 71.9***
al(900)	*900*	*300*	*1629 ± 24 / 9.2*	*199.3 ± 3.2 / 15.2*	*54.3 ± 2.0 /143.3*	***53.4 ± 2.6 / 101.6***
al(1000)	*1000*	*333*	*1839 ± 28 / 12.9*	*223.3 ± 3.8 / 21.7*	*60.7 ± 2.3 / 199*	***58.1 ± 2.2 / 143***

Con respecto a una solución válida sin robustecer, la solución obtenida mediante el glotón reduce dramáticamente la rigidez entre 4 y 8 veces, según el tamaño de la instancia. El GRASP 1 puede reducir la rigidez del glotón a la mitad o un tercio con una pequeña inversión extra, y el GRASP 2 encuentra rigideces aún menores consumiendo menos tiempo de máquina. Sin lugar a dudas, las mejoras propuestas en este capítulo han hecho diferencias en la calidad y eficiencia del GRASP.

Punto de Mejora 5: Reciclar los elementos del viejo *RefSet* en el nuevo *RefSet*. Originalmente parecía una buena idea: concentrémonos en las mejores soluciones presentes, olvidemos el pasado, porque el objetivo es encontrar una solución mejorada en la próxima generación. Como idea es buena, pero la práctica demuestra que en 1 o 2 de cada 10 generaciones no solo no se puede encontrar una mejor solución, sino que las soluciones hijas son de menor calidad que sus padres. El único modo de garantizar que siempre tenemos las mejores soluciones es incluir todos los mejores elementos anteriores.

Pseudo código Mejorado de Búsqueda Dispersa para Coloración Robusta.

{Paso 1: GRASP para Generación de Soluciones.}
DESDE *i=1* a *100*
Generar una Solución Glotona Mejorada (Ver Mejora 1)
Sea κ=3. Usar en ella el Método de Búsqueda Local (Ver Mejora 2)
Sea κ=1. Usar en ella el Método de Búsqueda Local
Sea κ=3. Usar en ella el Método de Búsqueda Local
Sea κ=1. Usar en ella el Método de Búsqueda Local (Mejoras 3 y 4)
Incluir todas las soluciones mejoradas en el *BigSet*.
FIN {Desde}

{Paso 2: Método de actualización del RefSet}
El conjunto de referencia deberá tener entre 5 y 20 elementos.
Agregar los elementos del *RefSet* al *BigSet*. (Mejora 5)
Eliminar del *BigSet* todas las soluciones redundantes.

Seleccionar 50% de soluciones con coloración válida y mínima rigidez.
Seleccionar el otro 50% de las soluciones con uno o dos vértices mal coloreados y mínima rigidez.
Vaciar *BigSet*.

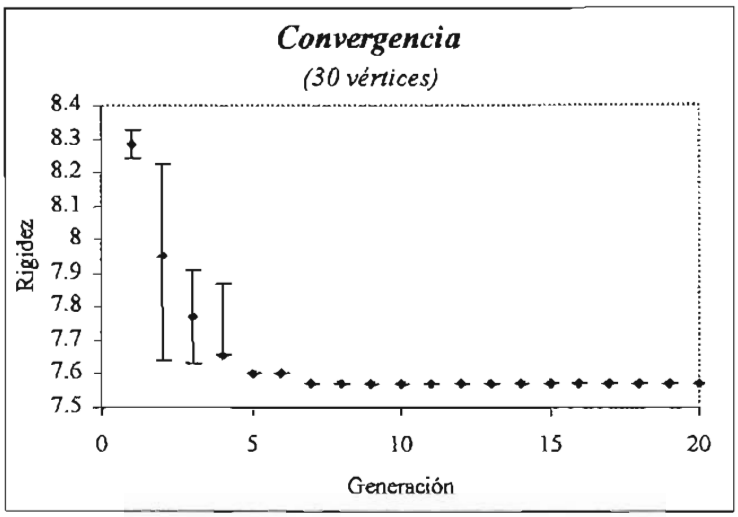
{Paso 3: Método de Generación de Subconjuntos}
Considerar todas las combinaciones posibles de conjuntos de dos elementos.
Considerar la mezcla de los mejores 4 elementos en los conjuntos de 3 elementos.

{Paso 4: Método de Combinación de Soluciones}
Para todos los subconjuntos de dos elementos *a* y *b*
  Desde *m=1* hasta *Partitions*
    Mezclar aleatoriamente los elementos de la solución *col(b)* en *col(a)* (ver Sección 4.5)
    Usar el método de mejora en la nueva solución
    Agregar la solución al *BigSet*
  Fin {Desde}
Fin {Para todos}
Para los cuatro mejores elementos del *RefSet*
  Mezclar los tres elementos en una proporción de 1/3 cada uno
  Usar en las soluciones el método de mejora y agregarlas al *BigSet*
Fin {Para cuatro}
Actualizar el *RefSet* con las mejores soluciones del *BigSet*
Si por lo menos hemos encontrado una mejor solución en el *RefSet* de esta generación, regresar al paso 2, si no, terminar el algoritmo.

5.2. Convergencia de soluciones.

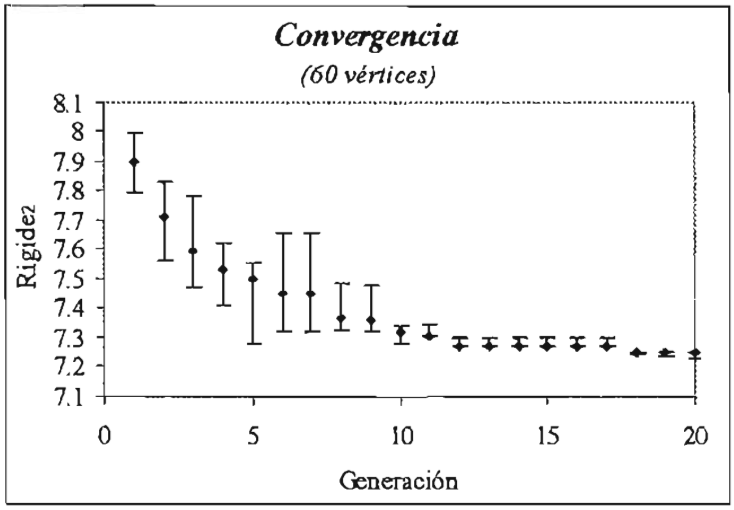
Se presentan a continuación, a manera de ejemplo, cuatro curvas donde se muestra el comportamiento del algoritmo de Búsqueda Dispersa con respecto al tamaño de instancia para 30, 60, 90 y 120 vértices. En todas estas instancias el tamaño del conjunto inicial (la solución GRASP) son 100 elementos, con un conjunto de referencia de 5 elementos y 5 pasos entre soluciones, y una proporción de 40/60 entre calidad y diversidad[2]. Se tomaron diez mediciones por cada generación. El punto muestra la mediana, la barra inferior marca el primer cuartil y la superior el tercero.

[2] Como solamente tenemos 5 elementos, tomar 50/50 no tiene sentido. De esta forma tomamos dos elementos de calidad y tres de diversidad.



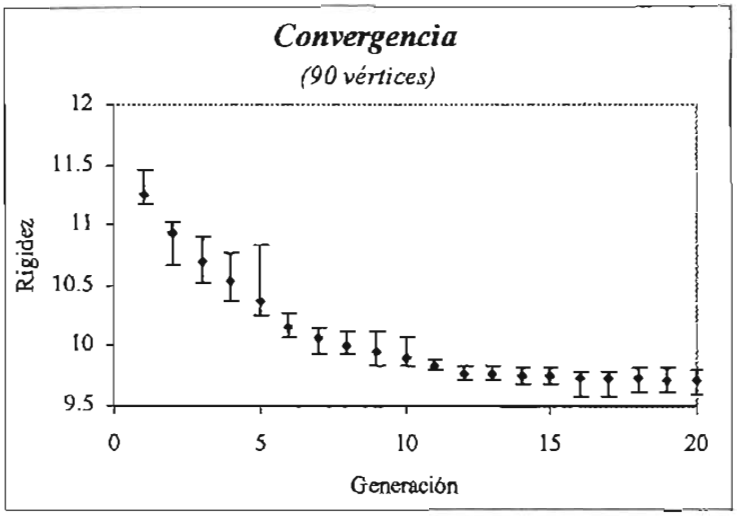
Gráfica 5.1. Comportamiento del algoritmo para un grafo de 30 vértices.

En la *gráfica 5.1.* si tomamos en cuenta que nuestro conjunto de referencia es bastante pequeño (5 elementos) podemos esperar una convergencia mas lenta, pero se puede observar que en tan solo 7 generaciones se llega a un solo mejor valor en la instancia de 30 vértices.

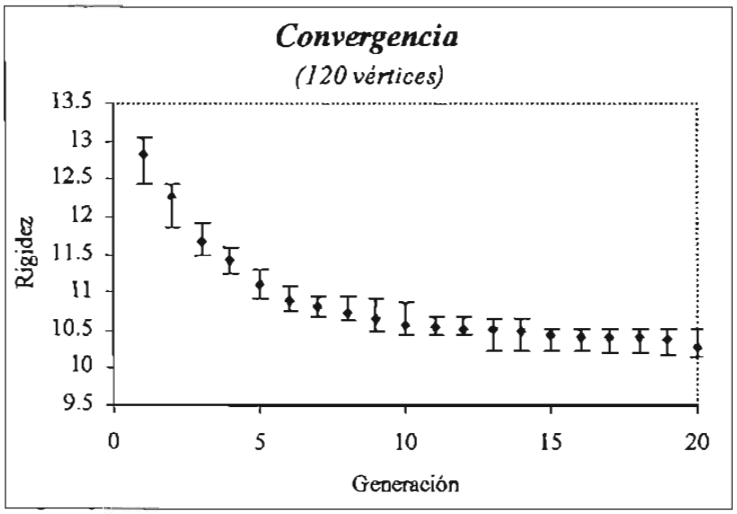


Gráfica 5.2. Comportamiento del algoritmo para un grafo de 60 vértices.

Se puede ver que para instancias de 60, 90 y 120 vértices la convergencia es un tanto mas lenta, aún así, las soluciones podría mejorar al permitir quizás 30 o mas generaciones, pero por cuestiones de rapidez, y ya que el beneficio en la calidad de la solución al duplicar el tiempo de ejecución es muy pequeño, se cortó a 20 generaciones.



Gráfica 5.3. Comportamiento del algoritmo para un grafo de 90 vértices.



Gráfica 5.4. Comportamiento del algoritmo para un grafo de 120 vértices.

5.3. Ejemplo paso a paso.

Para ilustrar los algoritmos desarrollados en esta tesis, se muestra a continuación cómo se resuelve con ellos una instancia del Problema de Coloración Robusta. Consideremos un grafo de nueve vértices, el cual será pintado con 5 colores (que es su número cromático). La matriz asociada a este grafo se presenta a continuación:

	a	b	c	d	e	f	g	h	i
a	0.0000	0.0000	1.0000	1.0000	1.0000	0.0000	0.0000	1.0000	0.0000
b	0.0850	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
c	0.0000	0.0000	0.0000	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000
d	0.0000	0.0000	0.0000	0.0000	0.0000	1.0000	0.0000	0.0000	1.0000
e	0.0000	0.0000	0.4261	0.8580	0.0000	0.0000	1.0000	1.0000	1.0000
f	0.5150	0.0000	0.0000	0.0000	0.7898	0.0000	0.0000	0.0000	1.0000
g	0.2629	0.0000	0.0000	0.9362	0.0000	0.6095	0.0000	0.0000	0.0000
h	0.0000	0.0000	0.0000	0.3085	0.0000	0.5995	0.7728	0.0000	0.0000
i	0.5604	0.0000	0.0000	0.0000	0.0000	0.0000	0.1919	0.7804	0.0000

En la diagonal superior de esta se encuentra la información del grafo original: si en la entrada *(j, k)* hay un uno, los vértices están conectados, si es cero, no. La diagonal inferior tiene la información del grafo complementario, donde si la entrada es diferente a cero, tenemos un arco con la penalización indicada. Es claro que si en esta matriz la entrada (j, k) es distinta de cero, la (k, j) es cero (es decir, si tenemos un arco en el grafo original, no existe en el complementario y viceversa).

Los pasos 1.1 y 1.2 muestran como se genera una solución GRASP. A partir de los mejores GRASP's del paso 1 generamos el primer *RefSet* que utilizaremos en los pasos subsecuentes.

Paso 1: GRASP.

Paso 1.1. Encontrar una solución glotona aleatorizada. Esencialmente, el glotón utilizado selecciona al azar un vértice, y prueba todos los colores posibles; de los colores que no causan incompatibilidades (o que genera el menor número de aristas incompatibles) se incluye el color que agrega menor rigidez. Debido a que la rigidez a cada paso depende de cuales vértices ya fueron coloreados, tendremos una coloración distinta según la secuencia de llenado, como tenemos *n!* secuencias de llenado[3], tendremos a lo más *n!* soluciones diferentes (porque algunas se podrían repetir), lo que nos da una diversidad de soluciones muy aceptable.

En la instancia determinemos primero una secuencia de coloreado de vértices de las 362,880 secuencias posibles (9!=362,880), tomemos *(i b f d c e g a h)*. Como cualquier color es válido y sin rigidez en el primer vértice[4], tomemos cualquiera al azar, por ejemplo, el 2. El segundo vértice por pintar será el *b*, ya que son los únicos vértices pintados, solo hay dos opciones: que se pinten con el mismo color, o con un color distinto: si se pintan con el mismo color, generamos una coloración parcial incompatible, ya que *i* y *b* están conectados por una arista en el grafo original, otra opción es pintarlos con colores diferentes, lo cual es la mejor opción, ya que genera una coloración parcial válida con una rigidez de cero todavía. Tomemos por ejemplo, el color 5 (en este paso, cualquier color distinto a 2 tiene el mismo efecto):

a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	Aristas incompatibles	Rigidez
X	2	X	X	X	X	X	X	2	1	0
X	**5**	**X**	**X**	**X**	**X**	**X**	**X**	**2**	**0**	**0**

La solución marcada en negritas es la mejor. Pasemos ahora al tercer vértice, *"f"* en nuestra secuencia. Tenemos esencialmente tres opciones: pintarlo con un dos, un cinco o cualquier otro color, digamos 3, las distintas opciones se presentan a continuación:

[3] Cuando el vector de coloración está vacío tenemos *n* vértices por escoger, al seleccionar el primero tenemos *n-1* vértices libres y así sucesivamente. Por ello tenemos *n!* diferentes secuencias de llenado.

[4] Requerimos al menos dos vértices pintados para poder hablar de alguna incompatibilidad entre ellos.

a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	Aristas incompatibles	Rigidez
X	5	X	X	X	5	X	X	2	1	0
X	5	X	X	X	2	X	X	2	1	0
X	**5**	**X**	**X**	**X**	**3**	**X**	**X**	**2**	**0**	**0**

Usar 2 o 5, generará una coloración incompatible, la mejor opción es usar 3. Pasemos ahora a pintar el cuarto vértice, *d*, y ahora tenemos 4 opciones, pintar con los tres colores utilizados o usar uno diferente, digamos, 1. Las opciones se presentan a continuación:

a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	Aristas incompatibles	Rigidez
X	5	X	2	X	3	X	X	2	1	0
X	5	X	3	X	3	X	X	2	1	0
X	5	X	5	X	3	X	X	2	1	0
X	**5**	**X**	**1**	**X**	**3**	**X**	**X**	**2**	**0**	**0**

La mejor opción es pintar de 1 el vértice *d*. Pintemos ahora el quinto vértice, *c*. Tenemos 5 opciones, tomando ahora cualquier color, incluso el 4 que no se había tomado anteriormente:

a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	Aristas incompatibles	Rigidez
X	5	1	2	X	3	X	X	2	1	0
X	5	2	2	X	3	X	X	2	1	0
X	5	3	2	X	3	X	X	2	1	0
X	**5**	**4**	**2**	**X**	**3**	**X**	**X**	**2**	**0**	**0**
X	5	5	2	X	3	X	X	2	1	0

La mejor opción es pintar el vértice *c* con el color 4. Ahora pintemos el sexto vértice, *e*, y las opciones posibles de pintado son las siguientes:

a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	Aristas incompatibles	Rigidez
X	5	4	2	1	3	X	X	2	0	.858
X	5	4	2	2	3	X	X	2	1	0
X	5	4	2	3	3	X	X	2	0	.7898
X	**5**	**4**	**2**	**4**	**3**	**X**	**X**	**2**	**0**	**.4261**
X	5	4	2	5	3	X	X	2	1	0

La mejor opción es la marcada en negrita, que es válida y nos da 0.4261 que es la menor rigidez. Pintemos el séptimo vértice, *g*, las posibilidades son:

a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	Aristas incompatibles	Rigidez
X	5	4	2	4	3	1	X	2	0	1.3623
X	**5**	**4**	**2**	**4**	**3**	**2**	**X**	**2**	**0**	**0.618**
X	5	4	2	4	3	3	X	2	0	1.0356
X	5	4	2	4	3	4	X	2	2	0.4261
X	5	4	2	4	3	5	X	2	1	0.4261

La mejor opción es pintar *g* con el color 2, ya que sigue siendo válida y tiene una rigidez 0.6180. el siguiente vértice por pintar es el *a*, y las alternativas son:

a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	Aristas incompatibles	Rigidez
1	5	4	2	4	3	2	X	2	1	0.6180
2	5	4	2	4	3	2	X	2	1	1.4413
3	5	4	2	4	3	2	X	2	0	1.1330
4	5	4	2	4	3	2	X	2	2	0.6180
5	**5**	**4**	**2**	**4**	**3**	**2**	**X**	**2**	**0**	**0.7030**

El vértice a con color 5 es la solución válida de menor rigidez, y es la que se toma. Finalmente nos queda por pintar el vértice h. Las opciones son:

a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	Aristas incompatibles	Rigidez
5	**5**	**4**	**2**	**4**	**3**	**2**	**1**	**2**	**0**	**1.0115**
5	5	4	2	4	3	2	2	2	0	2.2562
5	5	4	2	4	3	2	3	2	0	1.3025
5	5	4	2	4	3	2	4	2	2	0.7030
5	5	4	2	4	3	2	5	2	2	0.7030

Tenemos finalmente una coloración válida, *(5 5 4 2 4 3 2 1 2)*, con una rigidez de 1.0115. Esta coloración es, de hecho, el valor óptimo de esta instancia[5].

Paso 1.2. Aplicar el método de mejora. Como en el método glotón ya alcanzamos el valor óptimo, no tiene sentido aplicar el método de mejora en esta solución. Tomemos una solución alternativa que puede dar nuestro glotón: *(3 5 2 4 2 1 3 4 3)* que es una coloración válida con una rigidez de *1.7498*.

"Calentando" la solución. En esta fase dada una coloración, primero la "calentamos" al permitir algunas aristas incompatibles y después la "enfriamos" hasta alcanzar una solución factible. En esta fase definimos una especie de función objetivo, que es la que buscamos minimizar:

$$f(c) = I(c) + \kappa \; R(c)$$

[5] Esto ocurrió debido a que la instancia, para efectos didácticos, se tomo con solo nueve vértices. Con este algoritmo glotón, al ejecutarlo varias veces, se suele encontrar el óptimo en instancias de hasta 20 vértices, lo que nos muestra su buen desempeño.

En esta función, *R(c)* es la rigidez de la coloración *c*, *I(c)* es el número de aristas incompatibles y κ es un factor de ponderación. En nuestro ejemplo al "calentar" la coloración hacemos $\kappa=3$ (ponderamos tres veces mas la rigidez que las incompatibilidades). Empecemos a "calentar" nuestra solución. En la tabla de resultados debemos agregar la columna de *f(c)*. Tomemos nuevamente una cierta secuencia de mejora, por ejemplo, *(f i a e c g h d b)*: busquemos las 5 opciones para pintar el vértice *f* y veamos cual es la mejor opción de acuerdo a nuestra función *f(c)*:

a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	Aristas incompatibles	Rigidez	*f(c)*
3	**5**	**2**	**4**	**2**	**1**	**3**	**4**	**3**	**0**	**1.7498**	**5.2494**
3	5	2	4	2	2	3	4	3	1	2.5396	8.6188
3	5	2	4	2	3	3	4	3	1	2.8743	9.6229
3	5	2	4	2	4	3	4	3	1	2.3493	8.0479
3	5	2	4	2	5	3	4	3	1	1.7498	6.2494

En este vértice no encontramos una mejor solución, pasemos al vértice siguiente, el *i*:

a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	Aristas incompatibles	Rigidez	*f(c)*
3	**5**	**2**	**4**	**2**	**1**	**3**	**4**	**1**	**1**	**0.9975**	**3.9925**
3	5	2	4	2	1	3	4	2	2	0.9975	4.9925
3	5	2	4	2	1	3	4	3	0	1.7498	5.2494
3	5	2	4	2	1	3	4	4	1	1.7779	6.337
3	**5**	**2**	**4**	**2**	**1**	**3**	**4**	**5**	**1**	**0.9975**	**3.9925**

Tenemos una mejora, con una rigidez de 0.9975 a costo de tener una arista incompatible, podemos usar el color 1 o el 5; usemos el uno y pasemos al siguiente vértice, el *a*:

a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	Aristas incompatibles	Rigidez	*f(c)*
1	5	2	4	2	1	3	4	1	1	1.8100	6.4300
2	5	2	4	2	1	3	4	1	3	0.7346	5.2038
3	5	2	4	2	1	3	4	1	1	0.9975	3.9925
4	5	2	4	2	1	3	4	1	3	0.7346	5.2038
5	**5**	**2**	**4**	**2**	**1**	**3**	**4**	**1**	**1**	**0.8196**	**3.4588**

El siguiente vértice es *e*:

a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	Aristas incompatibles	Rigidez	*f(c)*
5	5	2	4	1	1	3	4	1	2	1.1833	5.5499
5	5	2	4	2	1	3	4	1	1	0.8196	3.4588
5	**5**	**2**	**4**	**3**	**1**	**3**	**4**	**1**	**2**	**0.3935**	**3.1805**
5	5	2	4	4	1	3	4	1	2	1.2515	5.7545
5	5	2	4	5	1	3	4	1	3	0.3935	4.1805

En *c*, *g*, *h*, *d*, y *b* ya no se logro ninguna mejora a la solución.

Se volvió a correr el algoritmo, ahora en la secuencia *(g i c b f a d e h)* y ya no se encuentra mejora alguna. Nos encontramos en un óptimo local de la solución "calentada". La coloración es *(5 5 2 4 3 1 3 4 1)* tiene dos aristas incompatibles y una rigidez de 0.3935. Enfriemos ahora la solución.

"Enfriando" la coloración. Con nuestra coloración "caliente" cambiamos el parámetro a $\kappa=1$, y volvemos a aplicar el proceso, escojamos ahora la secuencia azarosa *(g f e h a d c b i)*:

a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	Aristas incompatibles	Rigidez	*f(c)*
5	**5**	**2**	**4**	**3**	**1**	**1**	**4**	**1**	**1**	**1.1949**	**2.1949**
5	5	2	4	3	1	2	4	1	2	0.3935	2.3935
5	5	2	4	3	1	3	4	1	2	0.3995	2.3995
5	5	2	4	3	1	4	4	1	1	2.1015	3.1015
5	5	2	4	3	1	5	4	1	2	0.6564	2.6564

Con el proceso de enfriado, la solución inicial toma un valor de *f(c)* toma el valor de 2.3995, el cual es reducido a 2.1949 en la solución en negrita. Pasemos al siguente vértice, el *f*:

a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	Aristas incompatibles	Rigidez	*f(c)*
5	5	2	4	3	1	1	4	1	1	1.1949	2.1949
5	5	2	4	3	2	1	4	1	1	0.5854	1.5854
5	**5**	**2**	**4**	**3**	**3**	**1**	**4**	**1**	**0**	**1.3752**	**1.3752**
5	5	2	4	3	4	1	4	1	1	1.1849	2.1849
5	5	2	4	3	5	1	4	1	1	1.1004	2.1004

Esta última coloración ya es válida e incluso tiene una rigidez menor que nuestra solución glotona (1.7498). Sigamos con la arista siguiente, *e*:

a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	Aristas incompatibles	Rigidez	*f(c)*
5	5	2	4	1	3	1	4	1	2	0.5854	2.5854
5	**5**	**2**	**4**	**2**	**3**	**1**	**4**	**1**	**0**	**1.0115**	**1.0115**
5	5	2	4	3	3	1	4	1	0	1.3752	1.3752
5	5	2	4	4	3	1	4	1	1	1.4434	2.4434
5	5	2	4	5	3	1	4	1	2	0.5854	2.5854

Ya no encontramos mejoras en los vértices restantes *(h, a, d, c, b, i)*. La coloración encontrada dentro del método de mejora *(5 5 2 4 2 3 1 4 1)* es equivalente a la encontrada con el glotón, la cual, como ya se dijo, es el valor óptimo.

Paso 2: Mantenimiento del Conjunto de Referencia. Supongamos que deseamos generar un *RefSet* de 4 elementos, con 50/50 entre calidad y diversidad. Supongamos ahora que en 10 iteraciones del GRASP descrito anteriormente, obtuvimos las siguientes coloraciones:

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*Aristas incompatibles*	*Rigidez*
1	4	3	1	5	1	2	4	5	3	1	0.9975
2	1	3	2	5	2	4	1	5	3	1	0.9975
3	4	2	1	5	1	2	4	5	3	1	0.9975
4	3	2	4	2	1	5	1	3	4	4	0.0000
5	2	3	4	5	4	1	2	5	3	1	0.9975
6	4	2	2	5	1	3	4	5	1	2	0.5714
7	5	5	3	1	2	4	3	1	2	2	0.3935
8	3	4	5	1	5	2	3	1	4	1	0.9975
9	1	4	2	5	2	3	1	5	4	1	0.9975
10	3	5	2	4	2	1	3	4	3	0	1.7498

Si observamos con atención, las coloraciones primera y segunda son exactamente iguales cambiando el 1 por el 2, el 2 por el 4 y el 4 por el 1; las coloraciones 4 y 1 son casi equivalentes, solo se distinguen en la segunda entrada, pero ambas tienen una sola arista conflictiva y su rigidez es exactamente la misma. Un buen criterio de eliminación de soluciones es si comparten rigidez y aristas conflictivas: son o bien la misma solución o son muy parecidas. Con este criterio, para eliminar redundancia, nuestro *RefSet* queda como se muestra a continuación:

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*Aristas incompatibles*	*Rigidez*
1	4	3	1	5	1	2	4	5	3	1	0.9975
2	3	2	4	2	1	5	1	3	4	4	0.0000
3	4	2	2	5	1	3	4	5	1	2	0.5714
4	5	5	3	1	2	4	3	1	2	2	0.3935
5	3	5	2	4	2	1	3	4	3	0	1.7498

Como se nos pide 50/50 entre soluciones de calidad (dos coloraciones válidas y dos con un par de aristas conflictivas), de entrada, solamente tenemos una solución válida. Lo que se hace en estos casos es agregar coloraciones no validas hasta llenar el *RefSet*.

Vayamos alternando las soluciones: una válida, y otra con una o dos aristas conflictivas, de menor rigidez a mayor. Primero que nada, así eliminamos la segunda solución (cuatro aristas conflictivas es demasiado), quedando nuestro conjunto de referencia como se muestra a continuación:

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*Aristas incompatibles*	*Rigidez*
1	3	5	2	4	2	1	3	4	3	0	1.7498
2	5	5	3	1	2	4	3	1	2	2	0.3935
3	4	2	2	5	1	3	4	5	1	2	0.5714
4	4	3	1	5	1	2	4	5	3	1	0.9975

Este es, finalmente, nuestro conjunto de referencia depurado y listo para usarse en el religado de trayectorias.

Paso 3: Método de Generación de subconjuntos. Se toman todos los subconjuntos de dos elementos (1,2), (1,3), (1,4), (2,3), (2,4), (3,4), que son 6 en total. También se consideran todas las mezclas de los primeros cuatro elementos del *RefSet*, todos los elementos en este caso: (1,2,3), (1,2,4), (1,3,4), (2,3,4).

Paso 4: Método de Combinación de Soluciones. Por brevedad y sin pérdida de generalidad, solo mezclaremos los dos primeros elementos del *RefSet*, considerando tres pasos entre la solución inicial y la final.

Una vez más, necesitamos una secuencia de inserción de una coloración en otra. Supongamos que ahora dicha secuencia es *(f g a d e h b i c)*, para el primer paso, tomamos los tres primeros elementos y para el segundo los primeros seis. El primer paso entre la coloración 1 y 2 es:

	a	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***Aristas Incompatibles***	***Rigidez***
1	**3**	5	2	4	2	**1**	**3**	4	3	0	1.7498
2	5	**5**	**3**	**1**	**2**	4	3	**1**	**2**	2	0.3935
1+2	**3**	**5**	**3**	**1**	**2**	**1**	**3**	**1**	**2**	4	1.1709

Esta solución mezclada tiene cuatro aristas incompatibles y una rigidez de 1.1709, ahora debemos aplicar el método de mejora en ella. Primero buscamos templarla con $\kappa=1$, luego se calienta con $\kappa=3$ y se vuelve a enfriar con $\kappa=1$. Ya hemos descrito dos veces este camino, así que solo se presentaran las mejoras encontradas por cada vértice explorado. Seguimos la secuencia *(c i b h e d a g f)*:

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	Aristas incompatibles	Rigidez	*f(c)*
	3	5	3	1	2	1	3	1	2	4	1.1709	5.1709
c	3	5	**2**	1	2	1	3	1	2	3	1.5970	4.5970
i	3	5	**2**	1	2	1	3	1	**5**	2	1.5970	3.5970
b	3	**4**	**2**	1	2	1	3	1	**5**	1	1.5970	2.5970
h	3	**4**	**2**	1	2	1	3	**5**	**5**	1	1.4694	2.4694
e	3	**4**	**2**	1	**2**	1	3	**5**	**5**	1	1.4694	2.4694
d	3	**4**	**2**	**1**	**2**	1	3	**5**	**5**	1	1.4694	2.4694
a	**4**	**4**	**2**	**1**	**2**	1	3	**5**	**5**	1	1.2915	2.2915
g	**4**	**4**	**2**	**1**	**2**	1	**3**	**5**	**5**	1	1.2915	2.2915
f	**4**	**4**	**2**	**1**	**2**	**3**	**3**	**5**	**5**	0	1.9010	1.9010

Hagamos otra iteración en la secuencia *(h e d a f b c i g)*.

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	Aristas incompatibles	Rigidez	*f(c)*
	4	4	2	1	2	3	3	5	5	0	1.9010	1.9010
h	4	4	2	1	2	3	3	**1**	5	0	1.4291	1.4291
e	4	4	2	1	**2**	3	3	**1**	5	0	1.4291	1.4291
d	4	4	2	**1**	**2**	3	3	**1**	5	0	1.4291	1.4291
a	**4**	4	2	**1**	**2**	3	3	**1**	5	0	1.4291	1.4291
f	**4**	4	2	**1**	**2**	**3**	3	**1**	5	0	1.4291	1.4291
b	**4**	**4**	2	**1**	**2**	**3**	3	**1**	5	0	1.4291	1.4291
c	**4**	**4**	**2**	**1**	**2**	**3**	3	**1**	5	0	1.4291	1.4291
i	**4**	**4**	**2**	**1**	**2**	**3**	3	**1**	**5**	0	1.4291	1.4291
g	**4**	**4**	**2**	**1**	**2**	**3**	**5**	**1**	**5**	0	1.0115	1.0115

Y hemos llegado al óptimo, sin necesidad de recalentar y enfriar.

Con la siguiente mezcla de soluciones con la secuencia *(f g a d e h b i c)* tomamos los seis primeros elementos *(f g a d e h)* de la primera solución y los tres últimos *(b i c)* de la segunda:

	a	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***Aristas incompatibles***	***Rigidez***
1	**3**	5	2	**4**	**2**	**1**	**3**	**4**	3	0	1.7498
2	5	**5**	**3**	1	2	4	3	1	**2**	2	0.3935
1+2	**3**	**5**	**3**	**4**	**2**	**1**	**3**	**4**	**2**	3	0.5714

Así, enfriemos esta segunda coloración, en la secuencia *(a g h f i d c b e)* y $\kappa=1$

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	Aristas incompatibles	Rigidez	*f(c)*
	3	5	3	4	2	1	3	4	2	3	0.5714	3.5714
a	**3**	5	3	4	2	1	3	4	2	2	0.8235	2.8235
h	**3**	5	3	4	2	1	3	**4**	2	2	0.8235	2.8235
g	**3**	5	3	4	2	1	**1**	**4**	2	1	1.6957	2.6957
f	**3**	5	3	4	2	**5**	**1**	**4**	2	2	0.5714	2.5714
i	**3**	5	3	4	2	**5**	**1**	**4**	**2**	1	1.3237	2.3237
d	**3**	5	3	**4**	2	**5**	**1**	**4**	**2**	1	1.3237	2.3237
c	**3**	5	**3**	**4**	2	**5**	**1**	**4**	**2**	1	1.3237	2.3237
b	**3**	5	**3**	**4**	2	**5**	**1**	**4**	**2**	1	1.3237	2.3237
e	**3**	5	**3**	**4**	2	**5**	**1**	**4**	**2**	1	1.3237	2.3237

Hagamos otra iteración ahora con la secuencia azarosa *(c d a g h b i f e)*:

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	Aristas incompatibles	Rigidez	*f(c)*
	3	5	3	4	2	5	1	4	2	1	1.3237	2.3237
c	3	5	**3**	4	2	5	1	4	2	1	1.3237	2.3237
d	3	5	**3**	**4**	2	5	1	4	2	1	1.3237	2.3237
a	**5**	5	**3**	**4**	2	5	1	4	2	1	1.1004	2.1004
g	**5**	5	**3**	**4**	2	5	**1**	4	2	1	1.1004	2.1004
h	**5**	5	**3**	**4**	2	5	**1**	**4**	2	1	1.1004	2.1004
b	**5**	**2**	**3**	**4**	2	5	**1**	**4**	2	1	1.0154	2.0154
i	**5**	**2**	**3**	**4**	2	5	**1**	**4**	**2**	1	1.0154	2.0154
f	**5**	**2**	**3**	**4**	2	**5**	**1**	**4**	**2**	1	1.0154	2.0154
e	**5**	**2**	**3**	**4**	**3**	**5**	**1**	**4**	**2**	0	1.4415	1.4415

Si hacemos una sola iteración más sobre la arista *a* llegamos al óptimo:

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	Aristas incompatibles	Rigidez	*f(c)*
a	**2**	2	3	4	3	5	1	4	2	0	1.0115	1.0115

Tenemos 4 elementos en el *RefSet*, el cual genera 6 parejas diferentes con dos "pasos" (al partirlo en tres partes) entre soluciones. Si sumamos las triadas de los cuatro primeros elementos del *RefSet*, que generan 18 elementos, cada uno de los cuales es enfriado, calentado, y vuelto a enfriar, considerándose las tres soluciones parciales. Al incluir además los 4 elementos del *RefSet* de la generación anterior, tenemos un total de (6*2+4)*3+4=52 coloraciones, de las cuales se seleccionarán las cuatro mejores, dos coloraciones de calidad y dos de diversidad para generar el conjunto de referencia de segunda generación.

Una vez que hemos mejorado y explicado con un ejemplo corto los algoritmos realizados, la pregunta que surge es: ¿Qué tan buenos son los algoritmos desarrollados con respecto a lo que se ha hecho con anterioridad? Un estudio de su comportamiento respecto al estado del arte de la Coloración Robusta y su comportamiento para encontrar coloraciones mínimas es lo que encontraremos en el próximo capítulo.

Capítulo 6: Experiencia Computacional

La calidad de un algoritmo solo puede ser medida al comparar su desempeño con respecto a la experiencia previa, tanto en la calidad de las soluciones que da, como en velocidad de ejecución del mismo.

Tanto el GRASP como el Algoritmo de Búsqueda Dispersa desarrollados en este trabajo de investigación, fueron probados en dos tipos de instancias: las al(n) extendidas hasta 120 vértices; y las instancias DIMACS, ambas mostradas en el capítulo 1.

En este capitulo se presenta un sumario de experiencia computacional que se ha obtenido con el algoritmo desarrollado. En la sección 5.1. Se hace un comparativo con las instancias al(n) presentadas en [Ramírez, 2003] y con instancias similares entre 20 y 120 vértices. En la Sección 5.2. Se hace un comparativo del desempeño del algoritmo de Búsqueda Dispersa para encontrar coloraciones mínimas en las Instancias DIMACS.

6.1. Instancias Robustas *al(n)*

Instancias Robustas *al(n)* de 10 a 30 vértices.

A continuación se hace un comparativo del algoritmo de Búsqueda Dispersa respecto a las instancias propuestas en [Ramírez, 2003], las cuales tienen entre 10 y 30 vértices. Con base en la tabla de resultados de dicho trabajo, se agregaron los resultados obtenidos con el Algoritmo de Búsqueda Dispersa. El asterisco indica el valor óptimo obtenido con el modelo 0-1, y en enfatizado tenemos la mejor solución conocida actualmente.

Tabla 6.1. Comparativo del desempeño del algoritmo de Búsqueda Dispersa en Coloración Robusta con respecto a los otros algoritmos previamente utilizados para estas instancias.

Instancia			***Rigidez en coloración válida***				
$G_{n,0.5}$	n	k	*Enumeración Parcial*	*Búsqueda Tabú*	*Recocido Simulado*	*Modelo 0-1*	*Búsqueda Dispersa (Este Trabajo)*
al(10)	*10*	*4*	***4.2386****	***4.2386****	***4.2386****	***4.2386****	***4.2386****
		5	***1.8716****	***1.8716****	***1.8716****	***1.8716****	***1.8716****
al(11)	*11*	*4*	***3.4450****	***3.4450****	***3.4450****	***3.4450****	***3.4450****
		5	*2.4988*	***2.0227****	***2.0227****	***2.0227****	***2.0227****
al(12)	*12*	*4*	***5.0815****	***5.0815****	***5.0815****	***5.0815****	***5.0815****
		5	*3.8561*	***2.9041****	***2.9041****	***2.9041****	***2.9041****
al(13)	*13*	*5*	*4.8958*	***4.0440****	***4.0440****	***4.0440****	***4.0440****
		6	*2.7845*	***2.2765****	***2.2765****	***2.2765****	***2.2765****
al(14)	*14*	*5*	*6.4479*	*5.3659*	*5.9756*	***5.2836****	***5.2836****
		6	*3.7227*	***2.9387****	***2.9387****	***2.9387****	***2.9387****
al(15)	*15*	*5*	*8.3658*	*6.0683*	***5.6376****	***5.6376****	***5.6376****
		6	*5.0757*	***3.5592****	***3.5592****	***3.5592****	***3.5592****
al(20)	*20*	*7*	*8.6477*	*6.0236*	***4.9557***	---	***4.9557***
		8	*5.3147*	*3.9421*	***3.2285***	---	***3.2285***
al(25)	*25*	*9*	*7.0819*	*6.7866*	*5.5344*	---	***5.2584***
		10	*5.5987*	*4.8113*	*3.7923*	---	***3.7368***
al(30)	*30*	*10*	*10.5818*	*10.1346*	*7.6417*	---	***6.8881***
		11	*8.5675*	*7.4842*	***4.7623***	---	***4.7623***

Como se puede ver, el algoritmo de Búsqueda Dispersa desarrollado en este trabajo nos da, en todos los casos, los valores óptimos cuando estuvieron disponibles, o los mejores resultados conocidos para la instancia, cuando el optimo es desconocido. La Técnica de Búsqueda Dispersa es la mejor técnica que existe para el Problema de Coloración Robusta en cuanto a calidad de resultados en estas instancias pequeñas.

Instancias Robustas al(n) de 20 a 120 vértices.

Se generaron grafos equivalentes a los presentados en [Ramírez, 2003], que tienen entre 20 y 120 vértices. En la *tabla 6.2* se muestra la rigidez en una coloración válida y su tiempo de ejecución. En la columna de enumeración parcial no se muestra el tiempo de ejecución pero en las instancias más grandes es de menos de 20 segundos. En la columna de Búsqueda Tabú el tamaño de lista de movimientos prohibidos fue de 10, con 500 iteraciones. El GRASP se ejecutó 100 veces y se presenta la mejor solución encontrada. En el algoritmo de Recocido Simulado se considera $\alpha=0.93$ y un número de iteraciones en cada generación igual a $exp(2/\alpha^n)$, donde n es el número de generación. En el algoritmo de Búsqueda Dispersa, se tomaron 100 soluciones iniciales, *50/50* entre calidad y diversidad, *5* elementos en el *RefSet* y *5* pasos para el religado de trayectorias.

Tabla 6.2. Comparativo de desempeño de los algoritmos para instancias grandes

Instancia			***Rigidez en coloración válida / Tiempo de ejecución (segs)***				
$G_{n,0.5}$	n	k	*Enumeración Parcial*	*Búsqueda Tabú*	Mejor **GRASP** de 100 soluciones	*Recocido Simulado*	***Búsqueda Dispersa (Este Trabajo)***
al(20)	*20*	*7*	*7.6712*	*7.0970 /*	*7.1423 / 0.14*	***6.9046*** */ 3.86*	***6.9046*** */ 1.3*
		8	*6.4122*	*4.7710 /*	***4.6934*** */ 0.15*	*4.7109 / 3.05*	***4.6934*** */ 1.3*
al(30)	*30*	*10*	*11.2167*	*8.0623 /*	***7.5749*** */ 0.44*	***7.5749*** */ 2.70*	***7.5749*** */ 3.6*
		11	*8.8135*	*6.0565 /*	*5.9318 / 0.49*	*5.9318 / 4.11*	***5.8890*** */ 3.6*
al(40)	*40*	*14*	*13.3676*	*7.1709 / 15*	*7.3950 / 1.0*	***7.0837*** */ 6.47*	***7.0837*** */ 8.5*
		15	*10.4788*	*5.8173 / 14*	*6.3117 / 1.0*	***5.6708*** */ 6.14*	***5.6708*** */ 7.2*
al(50)	*50*	*17*	*13.0348*	*9.8259 / 33*	*8.9531 / 1.9*	*8.4552 / 9.11*	***8.2587*** */ 13*
		18	*11.9881*	*7.4966 / 33*	*7.1464 / 1.9*	*6.8232 / 9.31*	***6.7164*** */ 12*
al(60)	*60*	*20*	*16.4285*	*9.8331 / 69*	*9.9687 / 3.3*	***8.8676*** */ 7.0*	***8.8676*** */ 19*
		21	*13.6716*	*8.2181 / 69*	*8.1430 / 3.4*	*7.7431 / 12.8*	***7.2380*** */ 20*
al(70)	*70*	*24*	*15.4938*	*11.1307 / 128*	*11.2388 / 5.31*	*9.7195 / 12.2*	***9.2634*** */ 36*
		25	*16.8077*	*9.5478 / 128*	*9.2145 / 5.56*	*8.5979 / 14.5*	***7.7048*** */ 33*
al(80)	*80*	*27*	*17.9539*	*11.1946 / 218*	*11.7512 / 8.0*	*10.7772 / 18.4*	***9.9835*** */ 52*
		28	*15.8635*	*10.5845 / 219*	*10.2631 / 8.2*	*9.0361 / 21.22*	***8.5961*** */ 43*
al(90)	*90*	*30*	*21.4830*	*12.2832 / 350*	*13.4919 / 11.5*	*11.1629 / 19.9*	***10.8911*** */ 47*
		31	*20.5228*	*11.3699 /349*	*11.5060 / 11.9*	*11.0050 / 18.0*	***9.5008*** */ 89*
al(100)	*100*	*34*	*20.9084*	*12.1932 /544*	*12.8675 / 15.8*	*10.6124 / 22.2*	***10.0470*** */ 123*
		35	*20.3636*	*12.0650 / 885*	*11.1317 / 15.6*	*10.1099 / 30.1*	***9.4259*** */ 124*
al(110)	*110*	*37*	*26.6229*	----	*12.7681 / 20.7*	*11.9302 / 41.4*	***10.8463*** */ 149*
		38	*24.7310*	----	*11.6574 / 20.2*	*10.3908 / 39.3*	***9.9558*** */ 171*
al(120)	*120*	*40*	*27.5105*	----	*15.0014 / 26.8*	*12.2537 / 56.4*	***11.3507*** */ 190*
		41	*30.6818*	----	*13.5266 / 27.5*	*11.2683 / 43.6*	***10.1258*** */ 191*

Nuevamente, en todas las instancias el algoritmo que tiene un mejor desempeño es el de Búsqueda Dispersa, le sigue el de Recocido Simulado, y después con comportamientos similares, están el GRASP y el de Búsqueda Tabú, con la ventaja de que el GRASP es mucho mas rápido.

6.2. Instancias Mínimas DIMACS

Instancias DIMACS de 11 a 197 vértices.

Como se mencionó anteriormente, el Algoritmo Robusto generado en este trabajo puede resolver también el problema de coloración mínima, aunque no es un algoritmo especializado en él. Como una evaluación alternativa de su comportamiento, se decidió probar el algoritmo en las instancias clásicas del problema de coloración mínima: los grafos DIMACS (ver sección 1.1.). Para los valores de rigidez en el grafo complementario se tomaron valores aleatorios con distribución uniforme (0,1) como los utilizados en los grafos robustos *al(n)*.

Tabla 6.3. Desempeño del algoritmo de Búsqueda Dispersa en Coloración Robusta, en el problema extremo de coloración mínima en instancias DIMACS de menos de 200 vértices.

Instancia	*Número de Vértices*	*Densidad*	*Número Cromático* $\chi(G)$	*Solución factible con Búsqueda Dispersa*	*Colores extra para encontrar una coloración válida*
Mycielski-3	11	0.36	4	4	0
Mycielski-4	23	0.28	5	5	0
Mycielski-5	47	0.22	6	6	0
Mycielski-6	95	0.17	7	7	0
Mycielski-7	191	0.13	8	8	0
College Football	120	0.09	9	9	0
David Copperfield	87	0.11	11	11	0
Huckleberry Finn	74	0.11	11	11	0
Les Misèrables	80	0.08	10	10	0
Anna Karenina	138	0.05	11	11	0
Queen Graph 5x5	25	0.53	5	5	0
Queen Graph 6x6	36	0.46	7	7	0
Queen Graph 7x7	49	0.40	7	8	1
Queen Graph 8x8	64	0.36	9	10	1
Queen Graph 9x9	81	0.65	10	11	1
Queen Graph 10x10	100	0.59	≥ 10	13	≤ 3
Queen Graph 11x11	121	0.55	11	14	3
Queen Graph 12x12	144	0.50	≥ 12	15	≤ 3
Queen Graph 13x13	169	0.47	13	17	4
Queen Graph 14x14	196	0.44	≥ 14	18	≤ 4
Miles Graph 250	128	0.05	8	8	0
Miles Graph 500	128	0.14	20	20	0
Miles Graph 750	128	0.26	31	31	0
Miles Graph 1000	128	0.39	42	42	0
Miles Graph 1500	128	0.64	73	73	0
Mulsol.i.1	197	0.20	49	49	0
Mulsol.i.2	188	0.22	31	31	0
Mulsol.i.3	184	0.23	31	31	0
Mulsol.i.4	185	0.23	31	31	0
Mulsol.i.5	186	0.23	31	31	0

En la Tabla 6.3 se muestra un comparativo de las DIMACS utilizadas, el número de vértices en el grafo, su número cromático, su densidad, los colores requeridos para obtener una coloración válida con el algoritmo de Búsqueda Dispersa, y en la última columna se muestra el número de colores extra requeridos para obtener una coloración válida con el Algoritmo Robusto con respecto al valor mínimo.

Se puede observar que, aunque este algoritmo esta diseñado para resolver el problema de Coloración Robusta y no su caso especial de Coloración Mínima, su desempeño para encontrar coloraciones válidas en las instancias clásicas con menos de 200 vértices es bastante bueno, teniendo solo problemas con los grafos *Queen Graph*, que se caracterizan por su alta densidad y por el pequeño porcentaje entre cantidad de colores con respecto al número de vértices (entre 7 y 10% de número de colores respecto al número de vértices). Cabe mencionar que estos grafos son difíciles de resolver incluso para problemas especializados en coloración mínima, de los cuales a veces solo se conoce una cota al número cromático, y no el valor en si.

Instancias DIMACS (206 a 864 vértices)

Para ver el desempeño del algoritmo de Coloración Robusta en instancias muy grandes de Coloración Mínima, nuevamente se transformo el grafo de coloración mínima a uno de coloración robusta, agregando valores con distribución uniforme entre (0,1) en las aristas complementarias.

Tabla 6.4. Aplicado a instancias DIMACS de mas de 200 vértices.

Instancia	*Número de vértices*	*Densidad*	*Número Cromático* $\chi(G)$	*Solución factible con Búsqueda Dispersa*	*Colores extra para encontrar una coloración válida*
Zeroin.i.1	211	0.19	49	49	0
Zeroin.i.2	211	0.16	30	30	0
Zeroin.i.3	206	0.17	30	30	0
Class Scheduling	385	0.26	14	15	1
Class Scheduling nsh	352	0.24	14	15	1
Leighton 450-5a	450	0.06	5	11	6
Leighton 450-15a	450	0.08	15	19	4
Leighton 450-25a	450	0.08	25	26	1
Fpsol2.i.1	496	0.09	65	65	0
Fpsol2.i.2	451	0.09	30	30	0
Fpsol2.i.3	425	0.10	30	30	0
Inithx.i.1	864	0.05	54	54	0
Inithx.i.2	645	0.07	31	31	0
Inithx.i.3	621	0.07	31	31	0

Nuevamente el algoritmo de Búsqueda Dispersa tiene un comportamiento es muy bueno en instancias sobre problemas reales, como son *Zeroin, Fpsol* y *Inithx,* los cuales, como ya se mencionó, están basados en el problema de asignación de registros para variables en microprocesadores. Pero el algoritmo pierde eficiencia en las instancias *Leighton,* donde el porcentaje entre el número de colores con respecto al número de vértices es muy bajo, entre

1 y 5%. En este caso también nos enfrentamos a una familia muy difícil de colorear incluso para programas especializados en coloración mínima, cabe mencionar que los grafos *Leighton* de 5 colores ni siquiera se incluyen en los artículos especializados (solo los de 15 y 25 colores).

A lo largo de este documento de tesis ya hemos descrito en que consiste el Problema de Coloración Robusta, las Técnicas que se han utilizado para resolverlo, se han propuesto y desarrollado dos nuevos algoritmos, un GRASP que rápidamente da soluciones con calidad y uno de Búsqueda Dispersa que encuentra las mejores soluciones al día de hoy. Solamente resta a continuación dar las conclusiones y las aportaciones más importantes de este trabajo.

Conclusiones y Aportaciones Originales.

Conclusiones.

En este trabajo realizó una investigación sobre el Problema de Coloración Robusta: se explica en que consiste, sus características y las técnicas que se han utilizado para resolverlo. Se explican las técnicas GRASP y Búsqueda Dispersa. Se proponen tres algoritmos que resuelven el Problema de Coloración Robusta: uno es un Algoritmo de Búsqueda Dispersa, otro es un GRASP y finalmente, un algoritmo híbrido que utiliza la Técnica de Búsqueda Dispersa con soluciones iniciales generadas con GRASP. Estos algoritmos se programaron en *Visual Basic* y *Power Basic for Windows* y se usaron para resolver las instancias conocidas para el problema.

Se comparan los resultados obtenidos contra los obtenidos con algoritmos previos, y se encontró que el algoritmo Búsqueda Dispersa / GRASP desarrollado en este trabajo da las mejores soluciones encontradas a la fecha. El GRASP permite encontrar soluciones muy rápidamente de buena calidad.

Por otra parte, con los algoritmos se atacó el problema de coloración mínima utilizando para ello los grafos DIMACS, que son grafos de número cromático conocido, encontrándose en la gran mayoría de los casos que se podían obtener coloraciones válidas con el número cromático o utilizando un color extra únicamente, considerando que no es un algoritmo especializado en el problema, su desempeño es notablemente bueno.

Aportaciones Originales del Trabajo Doctoral.

- En esta investigación se propusieron tres algoritmos: un algoritmo de Búsqueda Dispersa puro, un algoritmo GRASP y un algoritmo híbrido Búsqueda Dispersa / GRASP. Los dos últimos son los que se desarrollan en este documento.

- Se desarrolló un programa en *VisualBasic 6.0* que resuelve el Problema de Coloración Robusta usando la Técnica de Búsqueda Dispersa. Este algoritmo encontraba las mejores soluciones respecto a las soluciones de trabajos anteriores, hasta ese momento. Debido a su lentitud y a su formato poco amigable, se decidió rescribir el código en *PowerBasic for Windows 7.0*. Este nuevo algoritmo, equivalente al original línea por línea, corría veinte veces mas rápido, con velocidades comparables a C++, y mucho mas amigablemente. Aunque se habla poco de esta propuesta en este documento, la referencia [Lara_ª, 2005] fue escrita fundamentada en este algoritmo.

- Se desarrolló en *PowerBasic for Windows 7.0* un programa que resuelven el Problema de Coloración Robusta mediante un GRASP. Este algoritmo obtiene mejores soluciones que otros algoritmos rápidos, como es el caso del algoritmo de Enumeración Parcial. De hecho su comportamiento es similar a los algoritmos de Busqueda Tabú y el de Recocido Simulado, con tiempos de ejecución menores.

- Se desarrolló en *PowerBasic for Windows 7.0* un programa que resuelve el Problema de Coloración Robusta con un híbrido Búsqueda Dispersa/GRASP. Como

ya se mencionó, este algoritmo encuentra las mejores soluciones conocidas a la fecha en las instancias propuestas en trabajos anteriores.

- Se amplió el estudio del comportamiento del Problema de Coloración Robusta. En estudios anteriores se incluyeron hasta instancias con 30 vértices. Este trabajo se proponen y resuelven instancias de hasta 120 vértices para el algoritmo de Búsqueda Dispersa / GRASP y un estudio que abarca desde 100 hasta 1000 vértices para el GRASP puro.

- Se extendió el uso del algoritmo al caso especial de Coloración Mínima, específicamente en las instancias DIMACS, encontrándose que a excepción de dos familias de grafos (diseñadas *ex profeso* para ser difíciles de colorear), en los demás casos, los cuales son aplicaciones reales, se encuentra una coloración con el número cromático, incluso en instancias de más de 800 vértices.

- Con base en el algoritmo de Búsqueda Dispersa puro, desarrollado en el año 2003, se creó el artículo *"Un Algoritmo Evolutivo para Resolver el Problema de Coloración Robusta"*, cuyos autores son *P. Lara Velázquez, M. A. Gutiérrez Andrade, J. Ramírez Rodríguez y J. López Bracho* por publicarse en la *Revista de Matemática: Teoría y Aplicaciones*, en el volumen 12, Números 1 & 2, del año 2005. Esta revista es editada por la *Universidad de Costa Rica*, está arbitrada y pertenece al índice de calidad de la Sociedad Europea Matemática *(Zentralblatt für Mathematick)* así como al *Current Statistical Index.*

- Con un modelo mejorado del algoritmo, se produjo el artículo: *Scatter Search in the Robust Coloring Problem.* Cuyos autores son *P. Lara y M. A. Gutiérrez* el cual fue enviado a la revista *Computación y Sistemas* del Instituto Politécnico Nacional. Cabe mencionar que este artículo se envió a esta revista ya que está incluida en el *Padrón de Excelencia de CONACYT.*

- Presenté la ponencia *"Scatter Search and Path Relinking in the Robust Coloring Problem"* cuyos autores son *P. Lara, M.A. Gutiérrez y S. de los Cobos*, en el *"INFORMS 2004 Annual Meeting"* en Denver, Colorado, EUA, el lunes 25 de Octubre de 2004 en la sesión MA03, donde estuvieron presentes Manuel Laguna, Rafael Martí y Vinicius Armentano.

- Se elaboró el artículo *"A Hybrid Algorithm for the Robust Coloring Problem"* Cuyos autores son *P. Lara, M. A. Gutiérrez y S. de los Cobos,* que fue enviado a la revista *Interfaces*, editada por el *Institute of Operations Research and Management Sciences*. Esta revista pertenece al *Science Citation Index* en la categoría de *Operations Research and Management Science.*

Actualmente se trabaja en una nueva investigación: generar un algoritmo híbrido *GRASP / Recocido Simulado* para el Problema de Coloración Robusta. Esta nueva investigación ya no tiene que ver ni con la Búsqueda Dispersa ni con otro Algoritmo Evolutivo, pero se menciona porque marca oficialmente el final de esta investigación doctoral.

Referencias

[Aiex, 2002] R.M. Aiex, M.G.C. Resende, and C.C. Ribeiro, *Probability Distribution of Solution Time in GRASP: An Experimental Investigation,* Journal of Heuristics, vol. 8, pp. 343-373, 2002

[Campos, 1999] V. Campos, M. Laguna and R. Martí. *Scatter Search for the Linear Ordering Problem New Ideas in Optimization,* D. Corne, M. Dorigo and F. Glover (Eds.), McGraw-Hill, pp. 331-339, 1999

[Campos, 2004] V. Campos, M. Laguna and R. Martí. *Context-Independent Scatter and Tabu Search for Permutation Problems.* Por publicarse en INFORMS Journal on Computing

[Cavique, 2001] Luís Cavique, César Rego, Isabel Themido. *A Scatter Search Algorithm for the Maximum Clique Problem.* Research Report. Hearin Center for Enterprise Science, University of Mississippi. 2001

[Diestel, 2000] Diestel, Reinhard. *Graph Theory.* Springer-Verlag. New York, 2000. (Electronic Edition).

[Eiben, 1998] A. E. Eiben, J. K. Van Der Hauw, J. I. Van Hemert. *Graph Coloring with Adaptative Evolutionary Algorithms.* Journal of Heuristics 4: 25-46, 1998

[Feo, 1995] T. A. Feo and MGC Resende. *Greedy Randomized Adaptative Search Procedures.* Journal of Global Optimization, 6 pags. 109-133, 1995.

[Galinier, 1999] Philippe Galinier, Jin-Kao Hao. *Hybrid Evolutionary Algorithms for Graph Coloring.* Journal of Combinatorial Optimization 3, 379-397, 1999.

[Glover, 1998] Glover, Fred. *A Template for Scatter Search and Path Relinking,* in *Artificial Evolution, Lecture Notes in Computer Science*, 1363, J.-K. Hao, E. Lutton, E. Ronald , M. Schoenauer and D. Snyers, Eds. Springer, pp. 13-54, 1998.

[Glover, 1999] Glover F. *Scatter Search and Path Relinking, in New Ideas in Optimization*, D. Corne, M. Dorigo and F. Glover, Eds., McGraw Hill, 1999, pp. 297-316.

[Glover, 2000] F. Glover, M. Laguna and R. Martí. *Fundamentals of Scatter Search and Path Relinking* Control and Cybernetics, vol. 39, no. 3 pp. 653-684 (2000)

[Glover, 2003] F. Glover, M. Laguna and R. Martí. *Scatter Search.* Advances in Evolutionary Computation: Theory and Applications, A. Ghosh and S. Tsutsui (Eds.), Springer-Verlag, New York, pp. 519-537 (2003)

[Glover_[a], 2004] F. Glover, M. Laguna and R. Martí. *New Ideas and Applications of Scatter Search and Path Relinking.* Por publicarse en New Optimization Techniques in Engineering, G. Onwubolu (Ed.), Springer-Verlag

[Glover_[b], 2004] F. Glover, M. Laguna and R. Martí. *Scatter Search and Path Relinking: Foundations and Advanced Designs.* Por publicarse en New Optimization Techniques in Engineering, G. Onwubolu (Ed.), Springer-Verlag

[Gutiérrez, 1991] M. A. Gutiérrez-Andrade. *La Técnica de Recocido Simulado y sus Aplicaciones.* Tesis De Doctorado. Universidad Nacional Autónoma de México, Facultad de Ingeniería. Agosto, 1991.

[Hamiez, 2001] Jean-Philippe Hamiez, Jin-Kao Hao. *Experimental Investigation of Scatter Search for Graph Coloring.* MIC'2001 - 4th Metaheuristics International Conference.

[Hamiez, 2002] Jean-Philippe Hamiez, Jin-Kao Hao. *Scatter Search for Graph Coloring.* Lecture Notes in Computer Science 2310: 168-179, Springer, 2002

[Laguna, 2002] M. Laguna. *Scatter Search.* In Handbook of Applied Optimization, P. M. Pardalos and M. G. C. Resende (Eds.), Oxford University Press, pp. 183-193 (2002)

[Laguna, 2001] M. Laguna and R. Martí. *A GRASP for Coloring Sparse Graphs.* Computational Optimization and Applications, vol. 19, no. 2, pp. 165-178 (2001)

[Laguna, 2004] M. Laguna and V. Armentano. *Lessons from Applying and Experimenting with Scatter Search.* To appear in Adaptive Memory and Evolution: Tabu Search and Scatter Search, Cesar Rego and Bahram Alidaee (Eds.)

[Lara, 2004] P. Lara, M. A. Gutiérrez. *Scatter Search in the Robust Coloring Problem.* Enviado a la Revista Computación y Sistemas (Mexico).

[Lara_[a], 2005] P. Lara-Velázquez, M. A. Gutiérrez-Andrade, J. Ramírez-Rodríguez, R. Lopez-Bracho. *Un Algoritmo Evolutivo para Resolver el Problema de Coloración Robusta.* Revista de Matemática: Teoría y Aplicaciones, vol. 12, nos. 1 & 2, pp. 111-119 (2005).

[Lara_[b], 2005] P. Lara-Velázquez, M. A. Gutiérrez-Andrade, S. de los Cobos. *A Hybrid Evolutive Algorithm for the Robust Coloring Problem.* Enviado a la revista Interfaces (EUA).

[Martí 2000] R. Martí, H. Lourenço and M. Laguna. *Assigning Proctors to Exams with Scatter Search Computing Tools for Modeling, Optimization and Simulation,* M. Laguna and J. L. Gonzalez Velarde (Eds.), Kluwer Academic Publishers, Boston, pp. 215-227 (2000)

[Martí, 2003] R. Martí, M. Laguna. Scatter Search: *Diseño Básico y Estrategias Avanzadas.* Revista Iberoamericana de Inteligencia Artificial, vol. 2, no. 4, pp. 123-130 (2003).

[Martí_[a] 2004] R. Martí, M. Laguna and V. Campos. *Scatter Search Vs. Genetic Algorithms: An Experimental Evaluation with Permutation Problems.* Por publicarse en Adaptive Memory and Evolution: Tabu Search and Scatter Search, Cesar Rego and Bahram Alidaee (Eds.)

[Martí_[b], 2004] R. Martí, M. Laguna, F. Glover. *Principles of Scatter Search.* Por publicarse en el European Journal of Operational Research

[McDiarmid, 2000] Colin McDiarmid and Bruce Reed. *Channel Assignment and Weighted Coloring*. Networks Vol. 36(2), 114-117 (2000).

[Murty, 1995] Katta G. Murty. *Operations Research: Deterministic Optimization Models.* Prentice-Hall. New Jersey, USA. 1995

[Ramírez, 2000] Javier Ramírez Rodríguez. *Extensiones del Problema de Coloración de Grafos.* Tesis de Doctorado. Universidad Complutense de Madrid. Facultad de Ciencias Matemáticas, Nov 2000

[Ramírez, 2003] J. Ramírez-Rodríguez, M. A. Gutiérrez-Andrade, R. López-Bracho, J. Yáñez-Gestoso. *Heuristics for the Robust Coloring Problem.* (En revisión).

[Ramírez, 2005] J. Ramírez-Rodríguez, R. López-Bracho, F. J. Zaragoza, J. Yáñez-Gestoso. *Bounds for the Robust Coloring Problem.* Resumen extendido para el evento: International Conference on Industrial Logistics (ICIL 2005). Febrero, 2005.

[Resende_[a], 2003] M. G. C. Resende C. C. Ribeiro. *GRASP and Path Relinking: Recent Advances and Applications.* Technical report, Internet and Network Systems Research Center, AT&T Labs Research, Florham Park, NJ, 2003.

[Resende_[b], 2003] M. G. C. Resende y J. L. González Velarde, GRASP: *Procedimientos de búsqueda miope aleatorizado y adaptatitvo,* Inteligencia Artificial, no. 19, pp. 61-76, 2003.

[Ribeiro, 1999] G. Ribeiro Filho, L. A. Nogueira Lorena. *Improvements on Constructive Genetic Approaches to Graph Coloring. IFORS'99* - The 15th Triennial Conference - The International Federation of Operational Research Societies. Beijing, China. 15-20/08/99.

[Yamashita, 2004] D. S. Yamashita, V. A. Armentano and M. Laguna. *Scatter Search for Project Scheduling with Resource Availability Costs.* Por publicarse en el European Journal of Operational Research

[Yañez, 2003] J. Yañez, J. Ramirez. *The Robust Coloring Problem.* European Journal of Operational Research. Vol. 148, No. 3, 2003, pp. 546-558.

Apéndice: Glosario de términos utilizados en este trabajo.

Búsqueda Local (Local Search). Es una técnica heurística que a partir de una solución inicial, realiza iterativamente una exploración alrededor de soluciones "vecinas" en busca de un valor mejorado. Una vecindad del número 2 en los reales podría ser el intervalo (1.99, 2.01); una vecindad de una solución del agente viajero podría ser permutar el lugar de dos ciudades en el recorrido.

Clan (Clique). Un clan es un subconjunto de vértices en un grafo los cuales generan un subgrafo completo entre si. Si este clan tiene r elementos, entonces para iluminar todo el grafo, requeriremos r o más elementos. Así, el clan más grande en el grafo nos da una cota mínima del número de colores a utilizar.

Coloración Robusta (Robust Coloring). Dados un grafo $G(V,E)$, $k>0$ un número de colores que permiten una coloración válida y una familia de penalizaciones de las aristas complementarias $\{p_{ij}, \{i,j\} \in \overline{E}\}$ el problema de Coloración Robusta consiste en determinar aquella k-coloración C_R^k con el menor grado de rigidez:

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

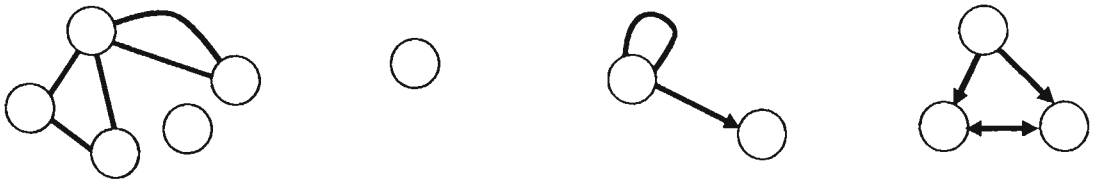
Dicho de otra forma, dado un grafo por colorear (no necesariamente con el mínimo de colores), existe un grafo complementario de aristas no incluidas en el grafo original y asociada a cada una de ellas, hay un cierto peso o penalización. Un grafo coloreado robustamente es aquel que satisface la coloración en el grafo original y minimiza el peso de las aristas con la misma cantidad de colores en el grafo complementario.

Coloración Válida (Valid Coloring). Dado un Grafo G, una coloración válida es un vector de coloración C^k que asocia un color c a cada vértice del grafo (c es un número entero entre *1* y *k)*, de tal forma que dos vértices adyacentes cualesquiera no comparten el mismo color. Al color asociado a un vértice i se le denota $C^k(i)$.

Densidad (Density). Es el porcentaje de aristas que tiene un grafo respecto a todas las aristas posibles. La densidad se podría definir como el cociente $2a/n(n-1)$ donde a es el número de aristas que tiene el grafo y n es el número de vértices.

Grado de un Vértice (Degree of a Vertex). Es el número de aristas incidentes en un vértice determinado.

Grafo (Graph). Un grafo es un par de subconjuntos $G=(V,E)$. Los elementos de E son pares del conjunto V. Los elementos de V se les denomina vértices (o nodos o puntos) y los elementos de E se les llaman aristas (o arcos o líneas).



(a) *(b)* *(c)* *(d)*

Figura A1.1. Cuatro ejemplos de Grafos

En la Figura A1.1. se presentan algunos ejemplos de grafos. En el grafo (a) tenemos 5 vértices, cuatro de ellos unidos por aristas, un vértice donde no inciden aristas y dos vértices unidos entre si por dos aristas. El grafo (b) consta únicamente de un vértice, sin aristas. En el grafo (c) tenemos una arista que empieza y termina en el mismo vértice, este tipo de arista se le llama un bucle; la otra arista esta dirigida. El grafo (d) tiene dos aristas dirigidas y una arista bidirigida.

Grafo Complementario (Complementary Graph). Dado un grafo *G(V,E)* se define como el grafo complementario al grafo $\overline{G}(V,\overline{E})$ donde $\overline{E}$ es el conjunto de aristas complementarias, es decir:

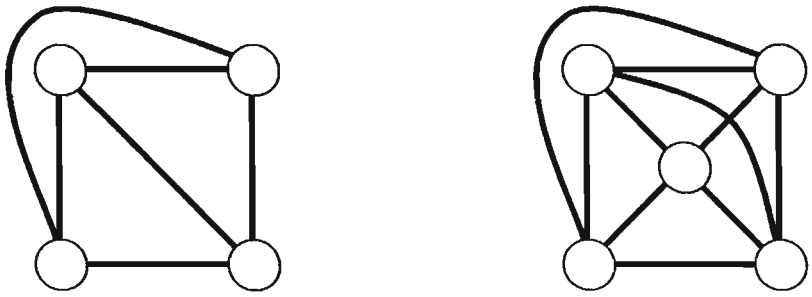
$$\{i,j\} \in \overline{E} \Leftrightarrow \{i,j\} \notin E$$

Grafo Completo (Complete Graph). Un grafo es completo cuando todas las aristas posibles en un grafo simple están incluidas en el conjunto *E*. En un grafo completo con *n* vértices, existen *n(n-1)/2* aristas.

Grafo Simple (Simple Graph). Un grafo es llamado simple si no tiene bucles y dos vértices cualesquiera están unidos a lo más por una arista. En este trabajo se utilizan grafos simples no dirigidos.

Vértices Adyacentes (Adjacent vertexes). Son cualesquiera dos vértices que están unidos por una arista.

Grafo Planar (Planar Graph). Un grafo es planar si se puede dibujar en un plano sin cruzar aristas. Como se puede ver en la figura A1.3., un grafo completo de cuatro vértices es un grafo planar; un grafo completo con cinco vértices ya no es planar, ya que siempre una de las aristas va a estar cruzada sobre alguna otra. Un grafo completo de cuatro vértices se puede colorear con cuatro colores, pero uno de cinco vértices requiere 5 colores. Así, el problema de los cuatro colores solo es válido en grafos planares.



(a) ***(b)***

Figura A1.2. Dos ejemplos de grafos completos para n=4 y n=5, El de la izquierda es planar, el de la derecha no.

Matriz de Adyacencia (Adjacency Matrix). Sea G un grafo simple, con n vértices etiquetados por $1,2,...n$ y los elementos $\{i, j\}$ las aristas que pertenecen a E. La *Matriz de Adyacencia* $A(G) = (a_{ij})_{n\times n}$ se define por:

$$a_{ij} := \begin{cases} 1 \ si \ \{i,j\} \in E \\ 0 \ en \ otro \ caso \end{cases}$$

En el caso de un grafo simple no orientado la matriz $A(G)$ es simétrica.

Número Cromático (Chromatic Number). Es el número mínimo de colores con el cual podemos colorear válidamente un grafo y se denota por $\chi(G)$. Para la figura 1.1 $\chi(G)=4$. Para la *figura 1.2b,* su número cromático es $\chi(G)=5$.

Problema del Agente Viajero (PAV) (Traveling Salesman Problem). Este problema es probablemente el mas conocido y estudiado dentro de la Optimización combinatoria. Dado un número de ciudades y dados los costos de viajar de una a otra, el PAV consiste en encontrar el viaje redondo mas económico que visita todas las ciudades y regresa a la ciudad inicial.

Rigidez de una coloración (Rigidity of a Coloring). Dados un grafo G y una familia conocida de penalizaciones $\{p_{ij}, \{i,j\} \in \overline{E}\}$ se define el grado de rigidez de la k-coloración C^k y se denota por $R(C^k)$ a la suma de las penalizaciones de las aristas complementarias cuyos extremos están igualmente coloreados:

$$R(C^k) = \sum_{\{i,j\}\in\overline{E}, C^k(i)=C^k(j)} p_{ij}$$