Universidad Autónoma Metropolitana
Azcapotzalco

División de Ciencias Básicas e Ingeniería
Licenciatura en Ingeniería en Computación

Proyecto terminal para obtener el grado de
Ingeniero en Computación presentado por:

Erika Morales Vega
205203524

Eduardo Alvarado Díaz
206204272

Trimestre 10 O

Fecha de entrega Diciembre 2010

Asesor
Dr. Pedro Lara Velázquez

UN ALGORITMO GRASP PARA EL PROBLEMA DE COLORACIÓN DIFUSA

Índice

Resumen

En este proyecto terminal se implementa un Algoritmo de Coloración Difusa basado en la tesis doctoral de Javier Ramírez (1) mediante el método GRASP que fue introducido por Feo y Resende (2). El proyecto terminal consta de tres bloques principales: generación de instancias aleatorias, implementación del GRASP y graficación de los números cromáticos difusos. La generación de instancias aleatorias se obtiene con dos distribuciones distintas: uniforme y triangular. El GRASP se divide en tres fases: construcción, mejora y actualización. La fase de construcción está dada por un algoritmo glotón, que obtiene una coloración válida no óptima. En las fases de mejora y actualización usamos un algoritmo de búsqueda local que trata de obtener una coloración válida óptima, si la solución obtenida es mejor a la almacenada, entra en la fase de actualización. Finalmente se realiza un análisis en la Función de densidad empírica para la variable aleatoria del número cromático difuso que se puede ver en las gráficas generadas.

Antecedentes

Grafo

Definición:

Un grafo (3) G consiste en un conjunto V de vértices (o nodos) y un conjunto E de aristas tales que cada arista $e \in E$ está asociado a un par no ordenado de vértices. Si una arista e está asociada a un único par de vértices v y w, se escribe $e = (v, w)$ o bien $e = (w, v)$. En este contexto, (v, w) denota la arista de un grafo no dirigido y no un par ordenado de números.

Definición:

Si $G = (V, E)$ es un grafo y $V = \{v_1,..., v_n\}$, se llama matriz de adyacencia (4) del grafo G y se denota como

$$Ma(G) = (e_{ij}) \; ; \; i,j \in \{1, \dots , n\}$$

Donde $e_{i,j}$ es el número de aristas cuyos extremos son $\{v_i, v_j\}$.

La matriz de adyacencia de un grafo es por lo tanto una matriz cuadrada de orden igual al número de vértices, simétrica y de enteros no negativos.

Coloración Difusa

A partir del concepto de número difuso y considerando que una función de coloración asigna números (colores) a los vértices de un grafo, es natural plantearse qué ocurre cuando el color asignado es un número difuso, introduciendo así la coloración difusa de un grafo nítido. También se puede plantear qué ocurre cuando la función de coloración es nítida pero el grafo es difuso; a partir de esta idea se plantea el problema de coloración difusa y se introduce el concepto de número cromático difuso.

Una de las formas que se puede representar un conjunto A es usando el concepto de función característica $\mu_A(\cdot)$ cuyo valor uno o cero indica si un elemento pertenece o no al conjunto A.

Al considerar que la función característica puede tomar cualquier valor en el intervalo [0,1], en 1965 Zadeh (5) introdujo el siguiente concepto de conjunto difuso, que generaliza al concepto clásico de conjunto:

Definición:

Dado un grafo $G = (V, E)$ y el conjunto finito de colores $\{1, 2, \ldots, c\}$, una coloración difusa es una aplicación

$C^c: V \rightarrow (\mu_{C^c}\ 1\ , \ldots, \mu_{C^c}\ c\ )$

Es decir, cada color k, $k \in \{1,\ldots c\}$ está presente en la coloración de un vértice $i \in V$ con una cierta intensidad $\mu_{C^c}\ k\ .$

Número Cromático

El número cromático (6) de G, denotado por el número mínimo k, para el cual existe una k-coloración que satisface lo siguiente: Para cualesquiera dos vértices $v_i, v_j \in V$ de G, tal que $\{v_i, v_j\} \in A$ tenemos que: $f\ v_i\ \neq f\ v_j\ .$

Número Cromático Difuso

Dado un grafo con aristas difusas $G = V, \mu$ el número cromático difuso de G estará formado por el conjunto difuso no normalizado cuyos elementos son las parejas formadas por el número cromático del $\alpha - corte\ G\ y\ \alpha_i$ con i que toma valores en $\{1,\ldots, p\}$.

No es difícil demostrar que el número cromático difuso es un conjunto convexo no normalizado cuyo valor modal está asociado con el grafo más restrictivo.

El problema de coloración difusa considera un grafo que en lugar de tener una matriz de adyacencia con aristas bien definidas (0,1), se definen de una forma difusa, la matriz tendrá números fraccionarios entre 0 y 1 (por ejemplo 0.1416 ó 0.9326); conservando la misma propiedad de simetría. Gracias a esto, los valores de muchos parámetros, por ejemplo densidad y número cromático, dejan de ser números y se convierten en variables aleatorias.

Algoritmo GRASP

La palabra GRASP (7) proviene de las siglas *Greedy Randomized Adaptive Search Procedures*, que en español podemos traducir como Procedimientos de Búsqueda basados en Funciones Voraces Aleatorizadas que se Adaptan.

A la función voraz, también se le conoce como función glotona. Esta función mide el beneficio de añadir cada uno de los elementos según la función objetivo y elegir la mejor. Notar que esta media no sabe qué ocurrirá en iteraciones sucesivas al realizar una elección, sino únicamente en esta iteración.

Se dice que el heurístico glotón se adapta porque en cada iteración se actualizan los beneficios obtenidos al añadir el elemento seleccionado a la solución parcial.

El heurístico es aleatorizado por que no selecciona el mejor candidato según la función glotona sino que, con el objeto de diversificar y no repetir soluciones en dos construcciones diferentes, se construye una lista con los mejores candidatos y entre ellos se toma uno al azar.

El término y los métodos GRASP fueron desarrollados al final de la década de los 80 por Feo y Resende (2) con el objetivo inicial de resolver problemas de cubrimientos de conjuntos. En 1995 fueron introducidas por estos mismos autores como una nueva técnica metaheurística[1] de propósito general.

GRASP es un procedimiento multi-arranque[2] en donde cada paso consiste en una fase de construcción y una de mejora. En la fase de construcción se aplica un procedimiento heurístico constructivo para obtener una buena solución inicial. Esta solución se mejora en la segunda fase mediante un algoritmo de búsqueda local. La mejor de todas las soluciones examinadas se guarda como resultado final.

[1] Algoritmos metaheurísticos o sencillamente heurísticos. Este término deriva de la palabra griega *heuriskein* que significa encontrar o descubrir y se usa en el ámbito de la optimización para describir una clase de algoritmos de resolución de problemas.

[2] También llamados Multi-Start o Re-Start, son procedimientos que básicamente almacenan la información relativa a soluciones ya generadas y la utilizan para la construcción de nuevas soluciones.

Justificación

Nuestro proyecto está enfocado a la implementación de un algoritmo GRASP con una eficiencia equivalente o mejor al algoritmo de recocido simulado que existe actualmente (8).

La relevancia de la implementación de este algoritmo es que resuelve las aplicaciones del problema de coloración difusa de grafos dentro de las cuales podemos encontrar (2):

- Problemas de planificación de tiempo y recursos, entre ellos, el problema de horarios de cursos de diferente duración, programación de exámenes, problema de coloración de aristas para modelos de planificación, y coloración de intervalos de grafos ponderados.
- Problemas de coloración mínima, por ejemplo, el problema de empaquetamiento o la versión del problema del agente viajero, determinación de conglomerados, y coloración de un mapa geográfico.

La originalidad del proyecto es que no se ha implementado un algoritmo para resolver el problema de Coloración Difusa, usando el método GRASP, y sólo ha quedado en investigación teórica.

Se necesitan conocimientos de programación, almacenamiento y recuperación de la información, estructuras de datos, diseño y análisis de algoritmos, probabilidad y estadística e inteligencia artificial, que forman parte del plan de estudios de la carrera de Ingeniería en Computación, por lo tanto, se necesita de un Ingeniero en Computación para realizarlo.

Este proyecto tiene a futuro, la posibilidad de servir como base para la implementación del algoritmo en otros lenguajes de alto nivel, o bien, implementarse para calendarizar mejor las UEA's dentro de nuestra universidad.

Problema de coloración difusa usando GRASP

Metodología

Para implementar el algoritmo de coloración difusa nos basamos en dividir el problema en tres bloques principales (ver Ilustración 1), los cuales se interrelacionan y se explican a continuación con más detalle dentro de cada módulo correspondiente.

Ilustración 1. Diagrama de bloques del algoritmo

Generación de instancias aleatorias

Este módulo consiste en generar las entradas del algoritmo GRASP, las cuales son archivos que contienen una matriz de adyacencia con las siguientes características:

- Cuadrada (n x n), de orden igual al número de vértices (n).
- Simétrica, ya que cada entrada a_{ij} es igual a a_{ji} y representa un vértice de un grafo no dirigido.
- La diagonal principal tiene costo cero, ya que no existe una arista al mismo vértice (no existen lazos o *loops*).
- El costo de las aristas está dado por números positivos entre [0,1) con cuatro decimales de precisión.

Utilizamos dos tipos de distribuciones para crear instancias y ver cuál es el comportamiento del algoritmo ─ la distribución uniforme y la distribución triangular.

Distribución uniforme

Usamos esta distribución ya que se caracteriza por el hecho de que todos los resultados posibles entre un cierto mínimo y máximo son igualmente probables (ver Ilustración 2).

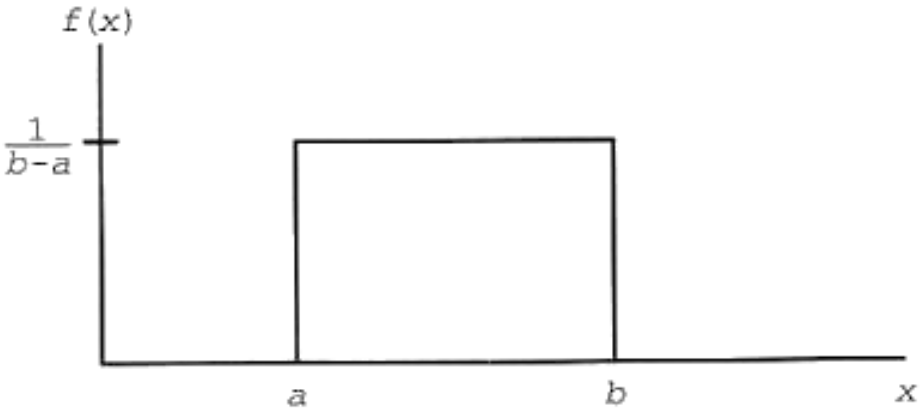

Ilustración 2. Distribución uniforme

Para generar los números aleatorios con distribución uniforme, usamos una función llamada *rnd*, que proviene del inglés *random* (aleatorio); y depende del lenguaje de programación su implementación ya que está soportada en la mayoría.

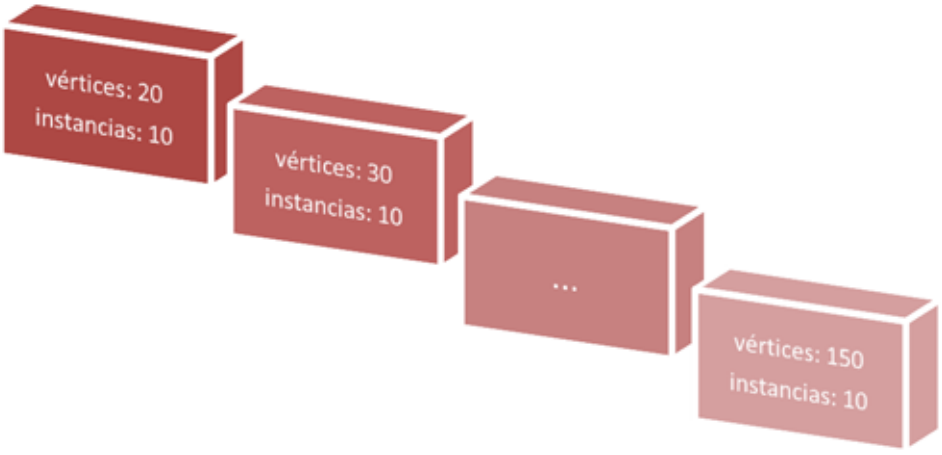

Ilustración 3. Secuencia de generación de instancias

Algoritmo 1. Pseudocódigo para el algoritmo de instancias aleatorias
Con distribución normal

```
1: vertices ← número de vértices
2: nombreArchivo ← nombre del archivo para escritura
3: numInstancias ← número de instancias a crear
4: PARA TODO número de instancias HACER
5:         Abrir "nombre de archivo + número de instancia" para escritura
6:         Memoria dinámica para mat(vertices,vertices)
7:         PARA TODO número de vértices i HACER
8:                 Elementos de la matriz en la diagonal principal mat(i,i) ← cero
9:                 PARA número de vértices delante de la diagonal principal j HACER
10:                        /* cálculo aleatorio con distribución normal */
11:                        mat(i,j) ← rnd
12:                        Reflejar el cálculo aleatorio en la posición simétrica mat(j,i)
13:                FIN PARA
14:        FIN PARA
15:        Escribir en archivo de salida número de vértices
16:        PARA TODO número de vértices i HACER
17:                PARA TODO número de vértices j HACER
18:                        Escribir en archivo de salida mat(i,j)
19:                FIN PARA
20:        FIN PARA
21:        Cerrar archivo de salida
22: FIN PARA
23: TERMINAR
```

Es importante decir que el algoritmo nos pregunta cuántos vértices tendrá cada instancia, y cuántas instancias se desean crear. En nuestro caso creamos 10 instancias para cada matriz de 20, 30,40,…, 150 vértices, teniendo en total 130 instancias con distribución uniforme (ver Ilustración 3).

Distribución triangular

Usamos esta distribución porque se usa comúnmente como una aproximación de otras distribuciones dado que depende de tres parámetros simples (el mínimo, el máximo, y el valor más probable). Puede tomar una variedad de formas, y es muy flexible para hacer modelos de una amplia variedad de supuestos (ver Ilustración 4). Otra característica que tiene es que es cerrada, eliminando la posibilidad de valores extremos que quizás podrían ocurrir en la realidad (9).

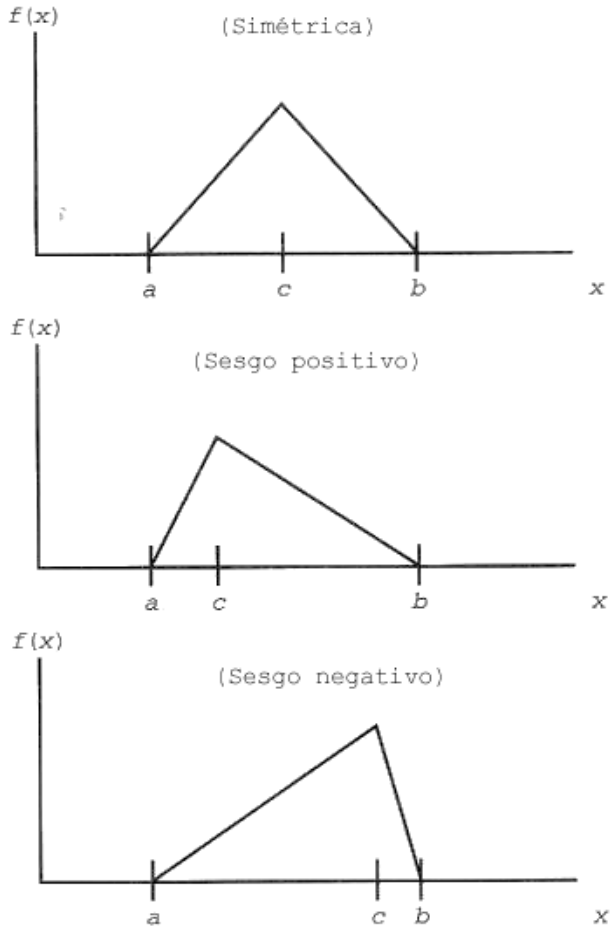

Ilustración 4. Distribución Triangular

Dependiendo del lenguaje, es como se implementa la instrucción para obtener números aleatorios, pero en nuestro caso, usamos una función 0.5*(*rnd*+*rnd)* para generar las instancias con distribución triangular.

Algoritmo 2. Pseudocódigo para el algoritmo de instancias aleatorias
Con distribución triangular

```
1: vertices ← número de vértices
2: nombreArchivo ← nombre del archivo para escritura
3: numInstancias ← número de instancias a crear
4: PARA TODO número de instancias HACER
5:         Abrir "nombre de archivo + número de instancia" para escritura
6:         Memoria dinámica para mat(vertices,vertices)
7:         PARA TODO número de vertices i HACER
8:                 Elementos de la matriz en la diagonal principal mat(i,i) ← cero
9:                 PARA número de vértices delante de la diagonal principal j HACER
10:                        /* cálculo aleatorio con distribución triangular */
11:                        mat(i,j) ← 0.5 * (rnd + rnd)
12:                        Reflejar el cálculo aleatorio en la posición simétrica mat(j,i)
13:                FIN PARA
14:        FIN PARA
15:        Escribir en archivo de salida número de vértices
16:        PARA TODO número de vertices i HACER
17:                PARA TODO número de vertices j HACER
18:                        Escribir en archivo de salida mat(i,j)
19:                FIN PARA
20:        FIN PARA
21:        Cerrar archivo de salida
22: FIN PARA
23: TERMINAR
```

Desarrollo del GRASP

Como hemos dicho, en este proyecto terminal implementamos el algoritmo GRASP para resolver el problema de coloración difusa. Dentro de un programa principal, tenemos un ciclo que aplica el algoritmo a cada instancia; se abre un archivo para lectura que contiene la matriz que se carga a memoria y se lleva el registro del tiempo de ejecución en un archivo para escritura.

Algoritmo 3. Pseudocódigo para el algoritmo de implementación del GRASP

Menú principal

```
1: nombreArchivo ← nombre del archivo para lectura
2: numInstancias ← número de instancias a leer
3: PARA TODO número de instancias HACER
4:         Abrir "nombre de archivo + número de instancia" para lectura
5:         Abrir "archivoTiempo + nombre de archivo" para agregar en escritura
6:         vertices ← número de vértices leido del archivo de lectura
7:         Memoria dinámica para matUm(vertices,vertices)
8:         PARA TODO número de vértices i HACER
9:                 PARA TODO número de vértices j HACER
10:                        matUm(i,j) ← leer de archivo de lectura valores de matriz
11:                FIN PARA
12:        FIN PARA
13:        Cerrar archivo de lectura
14:        tempo ← registro de tiempo antes de la subrutina
15:        Llamar a la subrutina GRASP
16:        Agregar en el archivo de escritura, tempo - tiempo después de la subrutina
17:        Cerrar archivo de agregar en escritura
18: FIN PARA
19: TERMINAR
```

La subrutina GRASP de manera general se aprecia en la Ilustración 5, y más adelante se explica detalladamente cada fase.

Mientras (Condición de parada)
Fase Constructiva
Seleccionar una lista de elementos candidatos.
Considerar una Lista Restringida de los mejores Candidatos
Seleccionar un elemento aleatoriamente de la Lista Restringida
Fase de Mejora
Realizar un proceso de búsqueda local a partir de la solución
Construida hasta que no se pueda mejorar más
Actualización
Si la solución obtenida mejora a la mejor almacenada, actualizarla.

Ilustración 5. Algoritmo GRASP

La subrutina tiene como objetivo recorrer un número de particiones sobre una matriz, llamados umbrales, en la que creará en memoria una matriz temporal de adyacencia, a la cual se le aplicará la fase constructiva y las fases de mejora y actualización. Como resultado nos da los números cromáticos difusos para cada umbral, guardándolos en un archivo para que más adelante se obtenga gráficamente la función de densidad.

Mientras más grande es el número de umbrales, mayor suavidad se tendrá en la gráfica de la función de densidad. En la subrutina del GRASP (Algoritmo 4) se muestra que tenemos un total de 1,000 umbrales, empezando desde 0.001 hasta 0.999.

El número de umbrales puede crecer hasta 10,000, debido a que manejamos una precisión de 4 decimales en los valores de la matriz, pero esto conlleva a tener un mayor tiempo de procesamiento, lo cual no es necesario ya que tenemos una buena aproximación con 1,000 umbrales. Si por el contrario disminuimos el número de umbrales a menos 100, tenemos una disminución de puntos los cuales no definen bien la función de densidad, es decir, no se aprecia la función de densidad bien definida.

Algoritmo 4. Pseudocódigo para el algoritmo de implementación del GRASP

```
                    Subrutina GRASP
1: Abrir "nombre de archivo + número de instancia" para escritura
2: trozos ← número de particiones, en este caso 1000
3: umbral ← 1 / trozos
4: incr ← umbral
5: PARA  número de trozos - 1 HACER
6:         PARA TODO  número de nodos i  HACER
7:                 colores(i) ← cero
8:                 visitados(i) ← cero
9:                 PARA TODO  número de vértices j  HACER
10:                        SI  matUm(i,j)  <=  umbral   Y   matUm(i,j)  >  0.0  ENTONCES
11:                                matTemp(i,j)  ← 1.0
12:                        SINO
13:                                matTemp(i,j)  ← 0.0
14:                        FIN SI
15:                FIN PARA
16:        FIN PARA
17:        Fase Constructiva
18:        Fases de Mejora y Actualización
19:        Escribir en archivo de salida número cromático óptimo
20:        umbral ← umbral + incr
21: FIN PARA
22: Cerrar archivo de escritura
23: TERMINAR
```

Para construir la matriz de adyacencia, nos basamos en los umbrales, los cuales indican el límite máximo. Si entra en la matriz de adyacencia toma un valor de 1, si no, el valor que tendrá será 0. Con la matriz de adyacencia es fácil ver cuáles son los vecinos de cada vértice.

Fase Constructiva

Ésta, es la primera fase donde literalmente se construye iterativamente una solución posible, considerando un elemento a cada paso. En cada iteración la elección del próximo elemento para ser añadido a la solución parcial viene determinada por una función glotona. Esta función mide el beneficio de añadir cada uno de los elementos según la función objetivo y elegir la mejor. Ésta medida no tiene en cuenta qué ocurrirá en iteraciones sucesivas al realizar una elección, sino únicamente en esta iteración.

Algoritmo 5. Pseudocódigo para el algoritmo de implementación del GRASP Glotón

```
1: PARA TODO número de vértices i HACER
2:         posición k ← vértice aleatorio
3:         MIENTRAS visitados(k) = 1 HACER
4:                 posición k ← vértice aleatorio
5:         FIN MIENTRAS
6:         visitados(k) ← 1
7:         xi ← 1
8:         HACER
9:                 bandera ← falso
10:                PARA TODO número de vértices j HACER
11:                        SI matTemp(k,j) = 1.0 Y colores(j) = xi ENTONCES
12:                                bandera ← verdadero
13:                        FIN SI
14:                FIN PARA
15:                SI bandera = verdadero ENTONCES
16:                        xi ← xi + 1
17:                FIN SI
18:        MIENTRAS bandera = 1
19:        colores (k) ← xi
20: FIN PARA
21: maximo ← colores(1)
22: PARA TODO número de vértices i HACER
23:        SI maximo < colores(i) ENTONCES
24:                maximo ← colores(i)
25:        FIN SI
26:        nuevaCol(i) ← colores(i)
27: FIN PARA
28: numCrom ← maximo
```

Fases de Mejora y Actualización

En ésta segunda fase se suele emplear un procedimiento de intercambio simple, con el objeto de no emplear mucho tiempo en esta mejora. Nótese que GRASP se basa en realizar múltiples iteraciones y quedarse con la mejor, por lo que no es especialmente útil para el método el detenerse demasiado en mejorar una solución dada. Para esto debemos tener un criterio de parada con un número de intentos.

En la última fase, sólo resta decir que si la solución obtenida mejora a la mejor almacenada, se actualiza y listo; si no la mejora, se desechan los cambios y continúa.

Algoritmo 6. Pseudocódigo para el algoritmo de implementación del GRASP
Búsqueda local

```
1: HACER
2:         bandera ← falso
3:         nuevoNumCrom ← numCrom - 1
4:         SI nuevoNumCrom > 0 ENTONCES
5:                 PARA TODO número de vértices i HACER
6:                         SI nuevaCol(i) = numCrom ENTONCES
7:                                 nuevaCol(i) ← nuevoNumCrom aleatorio
8:                         FIN SI
9:                 FIN PARA
10:                PARA TODO número de vértices i HACER
11:                        nuevaCol2(i) ← nuevaCol(i)
12:                FIN PARA
13:                numInc ← Calcular número de incompatibilidades en general
14:                numIntentos ← 10 * vertices * numCrom
15:                PARA TODO número de intentos w HACER
16:                        col_pru ← nuevoNumCrom aleatorio
17:                        ver_pru ← vértice aleatorio
18:                        nuevaCol2(ver_pru) ← col_pru
19:                        numInc2 ← Calcular número de incompatibilidades local
20:                        SI numInc >= numInc2 ENTONCES
21:                                nuevaCol(ver_pru) ← col_pru
22:                                numInc ← numInc2
23:                        SINO
24:                                nuevaCol2(ver_pru) ← nuevaCol(ver_pru)
25:                        FIN SI
26:                        SI numInc = 0 ENTONCES
27:                                PARA TODO número de vértices i HACER
28:                                        colores(i) ← nuevaCol(i)
29:                                FIN PARA
30:                                numCrom ← nuevoNumCrom
31:                                bandera ← verdadero
32:                                TERMINAR PARA
33:                        FIN SI
34:                FIN PARA
35:        FIN SI
36: MIENTRAS bandera = 1
```

Función de incompatibilidades

Dentro de la búsqueda local, tenemos la matriz de adyacencia y un arreglo de vértices, cada vértice tiene un número asignado (un color). Esta función tiene como objetivo llevar un conteo del número de incompatibilidades de manera general, que tiene un vértice con sus vértices adyacentes al tratar de disminuir el número cromático difuso.

Algoritmo 7. Pseudocódigo para el algoritmo del cálculo de incompatibilidades Generales

```
1: inc ← cero
2: PARA TODO número de vértices i HACER
3:         PARA número de vértices delante de la diagonal principal j HACER
4:                 SI matTemp(i,j) = 1.0 Y nuevaCol(i) = nuevaCol(j) ENTONCES
5:                         inc ← inc + 1
6:                 FIN SI
7:         FIN PARA
8: FIN PARA
9: Regresar número de incompatibilidades inc
10: TERMINAR
```

Dentro del procedimiento de intercambio simple, esta función tiene como objetivo llevar un conteo del número de incompatibilidades que tiene un vértice con sus vértices adyacentes al tratar de mejorar la coloración, esta función se hace de manera local con respecto al vértice que se le está cambiando el color.

Algoritmo 8. Pseudocódigo para el algoritmo del cálculo de incompatibilidades Locales

```
1: inc ← cero
2: PARA TODO número de vértices i HACER
3:         PARA número de vértices delante de la diagonal principal j HACER
4:                 SI matTemp(i,j) = 1.0 Y nuevaCol2(i) = nuevaCol2(j) ENTONCES
5:                         inc ← inc + 1
6:                 FIN SI
7:         FIN PARA
8: FIN PARA
9: Regresar número de incompatibilidades inc
10: TERMINAR
```

En la fase de actualización, si el número de incompatibilidades locales es menor que las generales, se actualiza y se acepta el cambio; si no la mejora, se desechan los cambios y continúa buscando mejorar la coloración para el siguiente umbral.

Distribución estadística del número cromático difuso del grafo 20_0

Mostraremos el funcionamiento general de la implementación del algoritmo de coloración difusa mediante el método GRASP para una instancia de 20 vértices. En la Tabla A1 se muestran los valores que se obtienen al generar una instancia de 20 vértices con una distribución uniforme (Algoritmo 1). Nótese que se cumple con las características antes mencionadas: es cuadrada, simétrica y con la diagonal principal en ceros.

A continuación se muestra que el control principal de la implementación (Algoritmo 3) toma la matriz de la Tabla A1, y construye la matriz de adyacencia para cada umbral. En la Tabla 1, se muestra la matriz de adyacencia para el umbral 0.1.

vértice	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	1
3	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
4	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0
8	1	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
9	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0
10	0	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0
14	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0
19	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 1. Matriz de adyacencia para el umbral 0.1

La subrutina GRASP (Algoritmo 4) entra en la fase de construcción que está dada por el algoritmo glotón (Algoritmo 5) que nos regresa como resultado una coloración válida

X = 3

2 2 3 2 2 1 1 1 3 3 1 2 1 1 2 1 1 2 1 1 (coloración válida)

Donde nos indica que la gráfica se puede colorear con 3 colores, y además nos da la lista de los colores que toma cada vértice. Para apreciarlo visualmente (ver Ilustración 6) y con fines prácticos le asignamos un color a cada número (Tabla 2) que sale del algoritmo glotón.

Número	***Color***	
1	Amarillo	
2	Rojo	
3	Azul	
4	Verde	
5	Morado	
6	Rosa	
7	Café	

Tabla 2. Convención de colores

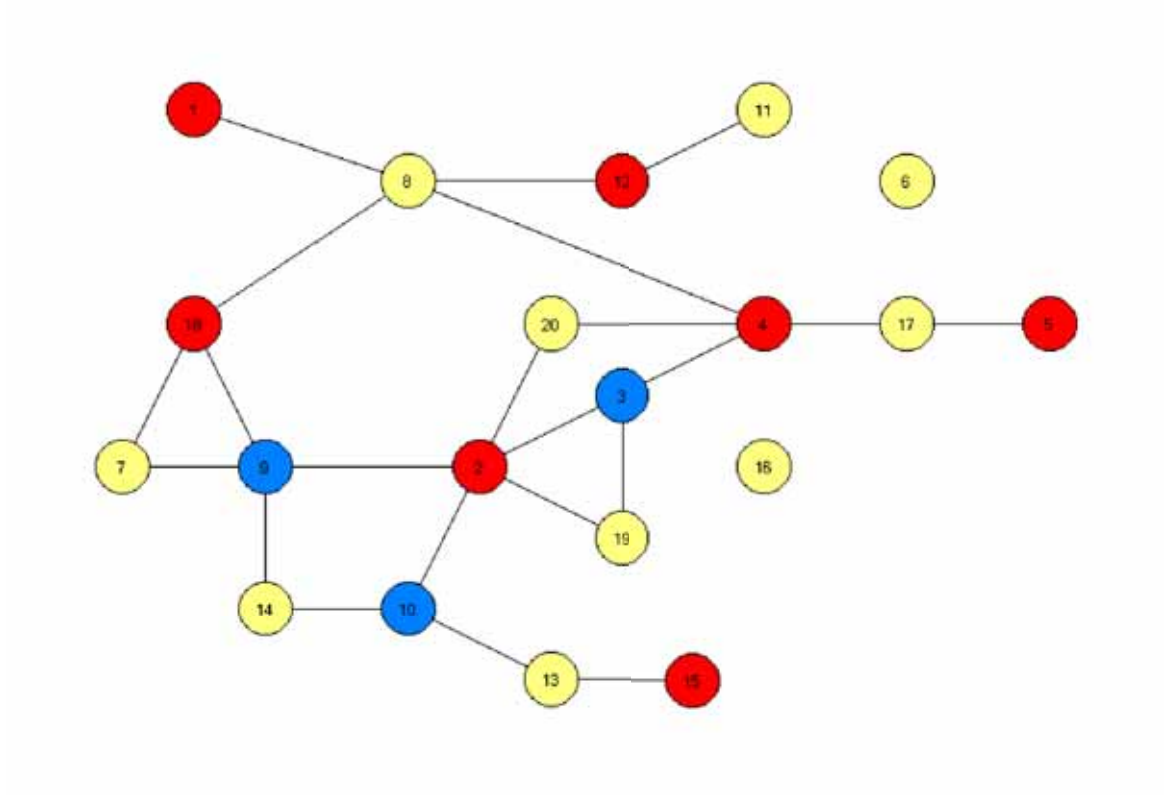

Ilustración 6. Coloración del glotón para el umbral 0.1

Entramos en la fase de mejora, para tratar de bajar el número cromático difuso a 2. Lo primero que hace el algoritmo de búsqueda local (Algoritmo 6) es cambiar los vértices que tienen el color azul, por un color aleatorio (ver Ilustración 7), al hacer dichos cambios surgen incompatibilidades que son calculadas por el algoritmo que calcula las incompatibilidades generales (Algoritmo 7), las cuales se muestran a continuación:

Xn= 2

2 2 **1** 2 2 1 1 1 **2** **2** 1 2 1 1 2 1 1 2 1 1 (coloración propuesta)

Incompatibilidad vértice-vértice
2 -> 9
2 -> 10
3 -> 19
9 -> 18

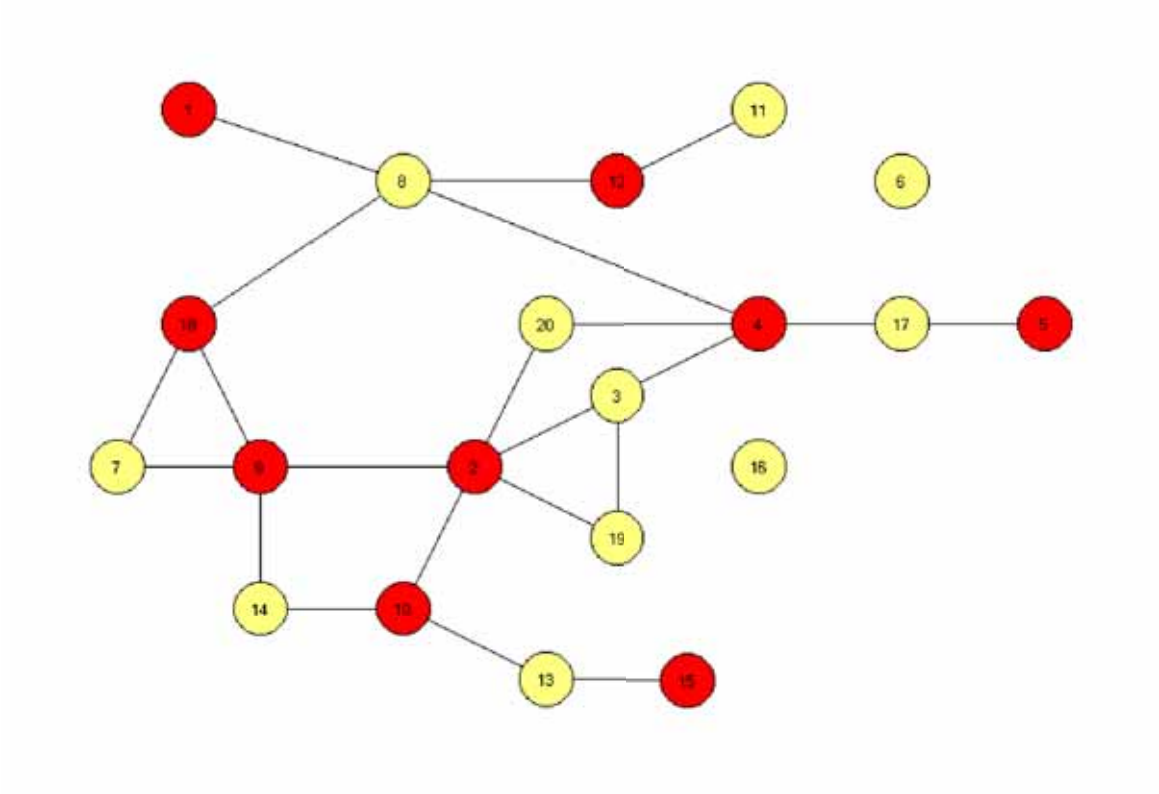

Ilustración 7. Coloración del GRASP con incompatibilidades para el umbral 0.1

El algoritmo entra en un ciclo con cierto número de intentos para lograr que el número de incompatibilidades generales sea cero, esto es, que se mejore la coloración. Después de varios intentos, no se pudo mejorar dicha coloración, se rechazan los cambios y nos quedamos con la coloración anterior (ver Ilustración 6).

Se puede apreciar en la Ilustración 7 que la gráfica se colorea con al menos 3 colores porque presenta triángulos, ejemplo: 2-3-19 y 7-9-18.

Para el umbral 0.2, la fase constructiva genera la siguiente matriz de adyacencia (Tabla 3).

vértice	vértice																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
2	1	0	1	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	1	1
3	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0
4	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	1
5	0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	1	0
6	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0
7	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	1	0	1	0	0
8	1	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0
9	0	1	1	0	0	0	1	0	0	1	0	0	0	1	0	0	0	1	0	0
10	0	1	0	0	1	1	0	0	1	0	0	0	1	1	0	0	1	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0
12	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0
13	0	0	0	1	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0
14	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0
15	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0
16	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	1	0
17	0	0	0	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0
18	0	0	0	0	0	0	1	1	1	0	0	0	0	1	0	1	0	0	0	0
19	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0
20	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 3. Matriz de adyacencia para el umbral 0.2

La subrutina GRASP (Algoritmo 4) entra en la fase de construcción mediante un glotón (Algoritmo 5), que nos regresa como resultado una coloración válida

X = 5

1 3 1 2 4 3 1 4 4 2 1 2 3 1 2 2 1 3 5 1 (coloración válida)

Donde nos indica que la gráfica se puede colorear con 5 colores (ver Ilustración 8).

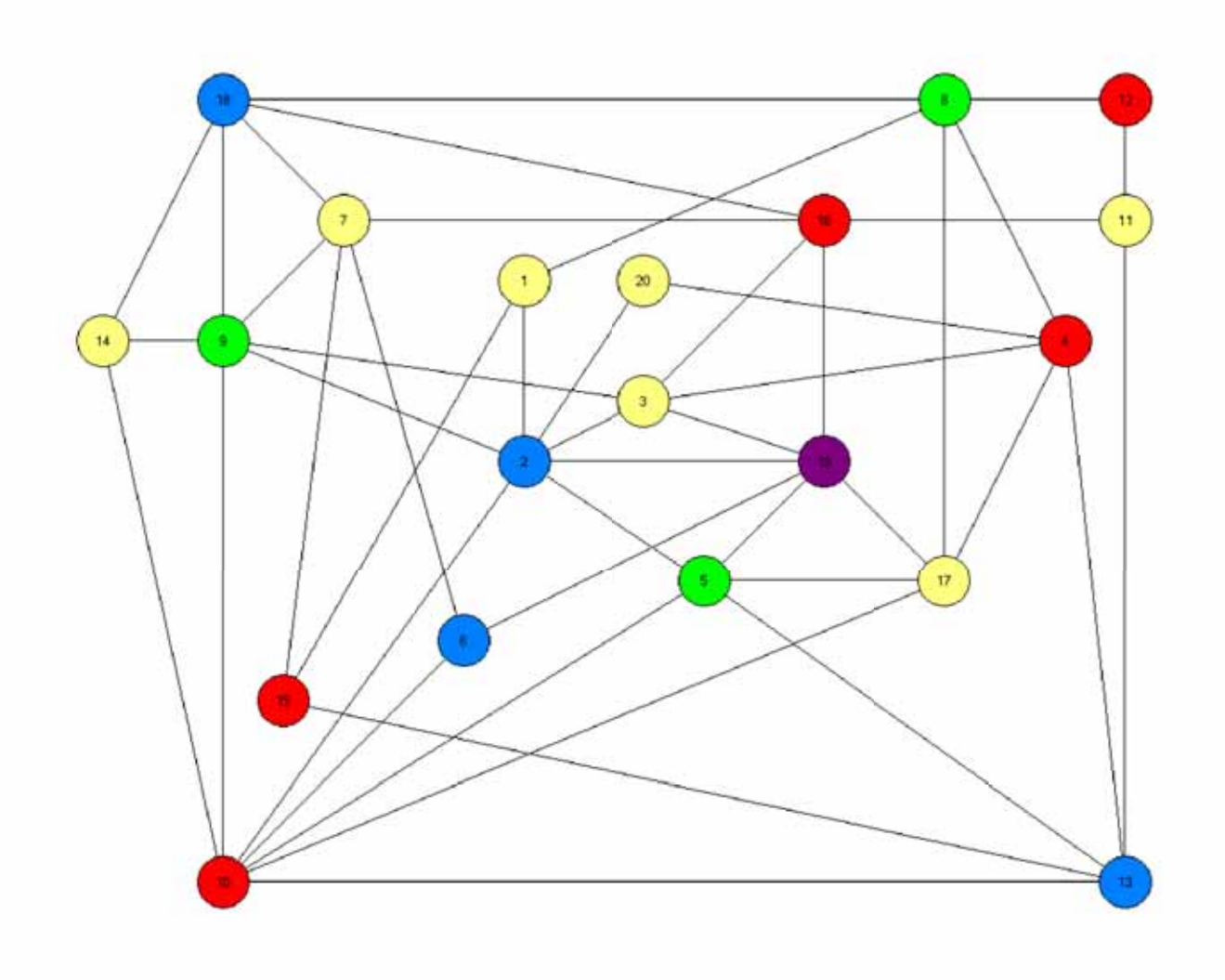

Ilustración 8. Coloración del glotón para el umbral 0.2

Entramos en la fase de mejora, para tratar de bajar el número cromático difuso a 4 colores, mediante el algoritmo de búsqueda local (Algoritmo 6). Lo primero que se hace es cambiar el vértice 14 que tienen color morado, por un color aleatorio (ver Ilustración 9), al hacer dichos cambios surgen incompatibilidades que son calculadas por el Algoritmo 7 (incompatibilidades generales), las cuales se muestran a continuación

Xn= 4

1 3 1 2 4 3 1 4 4 2 1 2 3 1 2 2 1 3 **1** 1 (coloración propuesta)

Incompatibilidad vértice-vértice
3 -> 19
17 -> 19

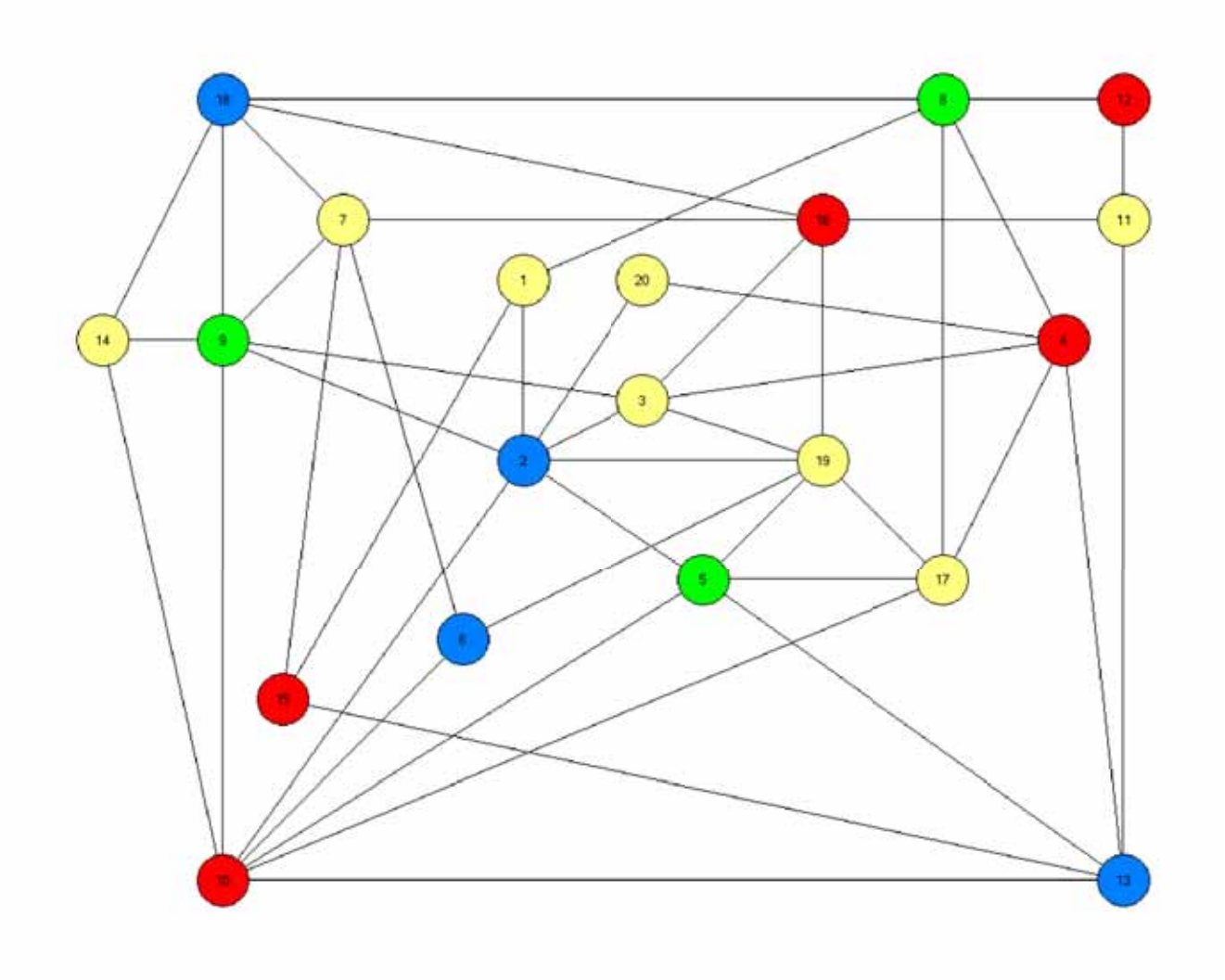

Ilustración 9. Coloración del GRASP con incompatibilidades en la fase de mejora para el umbral 0.2

El algoritmo entra en un ciclo para lograr que se mejore la coloración. Después de varios intentos, se logra mejorarla, y actualiza el número cromático difuso así como la coloración (ver Ilustración 10).

X = 4

1 3 1 2 4 **4** 1 4 4 2 **4** 2 3 1 2 **3** 1 **2** **2** 1 (nueva coloración)

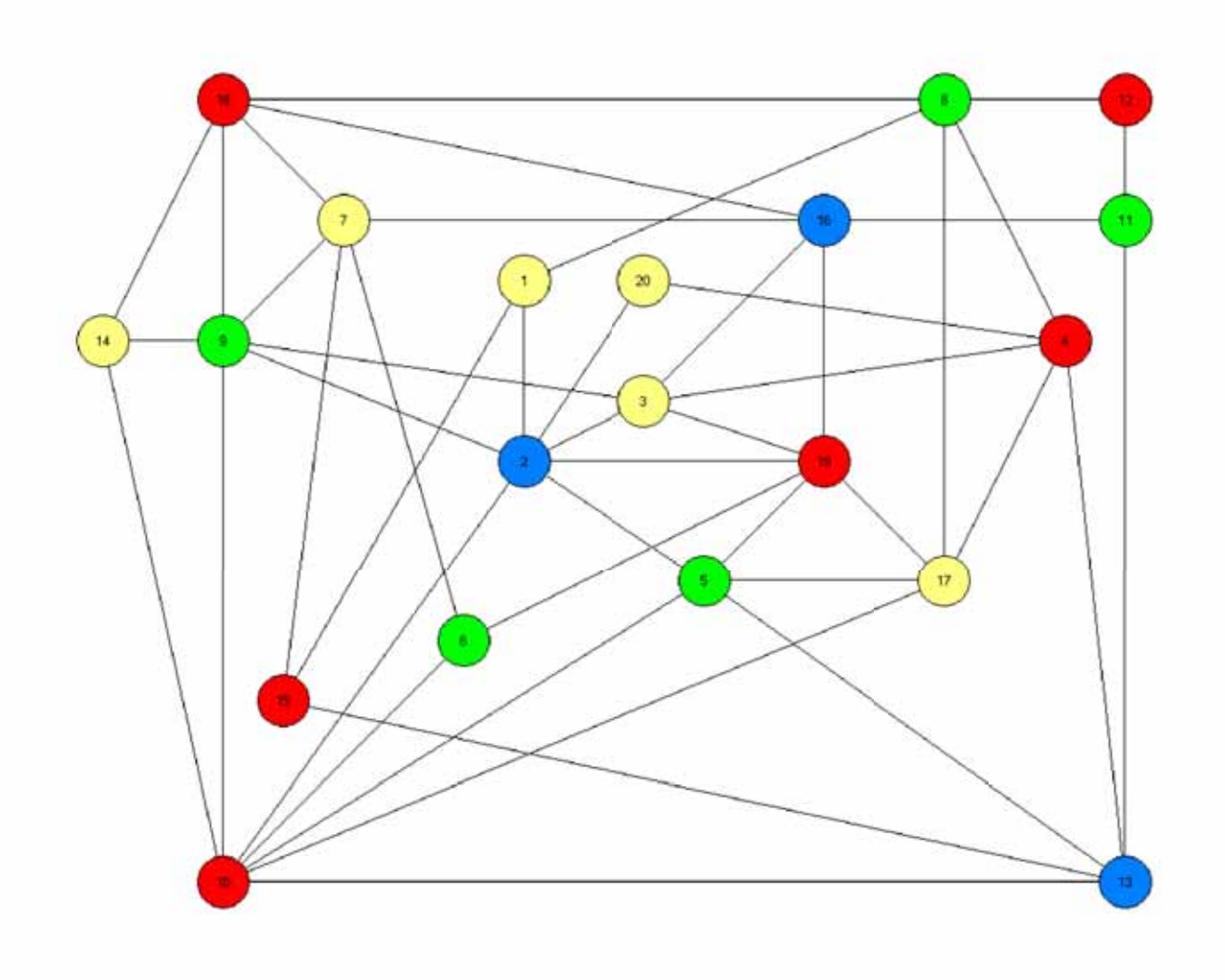

Ilustración 10. Coloración del GRASP en la fase de actualización para el umbral 0.2

Debemos observar que en la búsqueda local (Algoritmo 6), se elige un vértice al azar y se le asigna un color válido de forma aleatoria. Si el número de incompatibilidades locales es menor o igual que las incompatibilidades generales se acepta el cambio, de lo contrario se rechaza.

Nuevamente entra en la fase de mejora, pero ahora queremos disminuir el número cromático difuso a 3 colores. Se cambian los vértices verdes por colores aleatorios (ver Ilustración 11).

Xn= 3

1 3 1 2 **2** **1** 1 **3** **3** 2 **1** 2 3 1 2 3 1 2 2 1 (coloración propuesta)

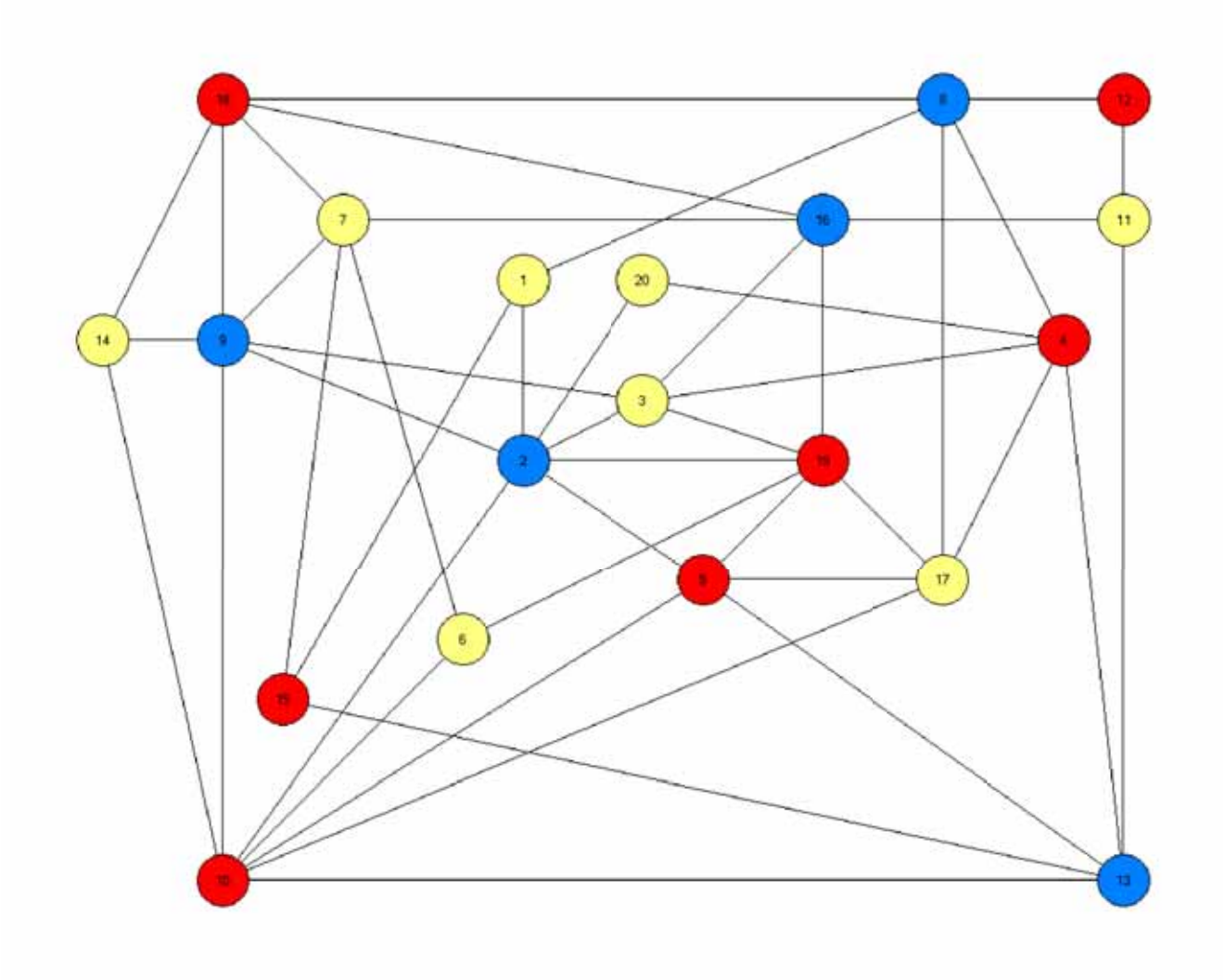

Ilustración 11. Coloración del GRASP con incompatibilidades en la fase de mejora para el umbral 0.2

Al hacer los cambios surgen incompatibilidades que son calculadas por el Algoritmo 7 (incompatibilidades generales), las cuales se muestran a continuación

Incompatibilidad vértice-vértice
2 -> 9
5 -> 10
5 -> 19
6 -> 7

Después de varios intentos, no se pudo mejorar dicha coloración, se rechazan los cambios y nos quedamos con la coloración anterior válida (ver Ilustración 10).

Finalmente mostraremos las coloraciones generadas para el umbral 0.3; en la fase constructiva se genera la siguiente matriz de adyacencia (Tabla 4).

vértice	vértice																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	1	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0
2	1	0	1	1	1	0	1	0	1	1	0	0	0	0	0	0	0	1	1	1
3	0	1	0	1	0	0	0	1	1	0	0	0	1	0	0	1	1	0	1	0
4	0	1	1	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	1
5	1	1	0	0	0	1	0	0	0	1	0	0	1	0	0	0	1	0	1	0
6	0	0	0	0	1	0	1	0	0	1	0	0	0	1	0	0	1	0	1	0
7	0	1	0	0	0	1	0	0	1	0	0	0	0	0	1	1	0	1	0	0
8	1	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0
9	0	1	1	0	0	0	1	0	0	1	0	0	0	1	0	0	0	1	0	0
10	0	1	0	0	1	1	0	0	1	0	0	0	1	1	1	0	1	1	0	0
11	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	1	0
12	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	0	0	0
13	0	0	1	1	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0
14	0	0	0	0	0	1	0	0	1	1	1	0	0	0	1	0	0	1	0	1
15	1	0	0	0	0	0	1	0	0	1	1	1	1	1	0	0	0	0	0	0
16	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	1	0
17	0	0	1	1	1	1	0	1	0	1	0	0	0	0	0	0	0	0	1	0
18	0	1	0	0	0	0	1	1	1	1	0	0	0	1	0	1	0	0	0	0
19	1	1	1	0	1	1	0	0	0	0	1	0	0	0	0	1	1	0	0	0
20	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

Tabla 4. Matriz de adyacencia para el umbral 0.3

La subrutina GRASP (Algoritmo 4) entra en la fase de construcción mediante un glotón (Algoritmo 5), que nos regresa como resultado una coloración válida

X = 7

1 4 1 3 6 4 1 4 2 5 4 1 7 1 2 2 2 3 3 2 (coloración válida)

Donde nos indica que la gráfica se puede colorear con 7 colores (ver Ilustración 12).

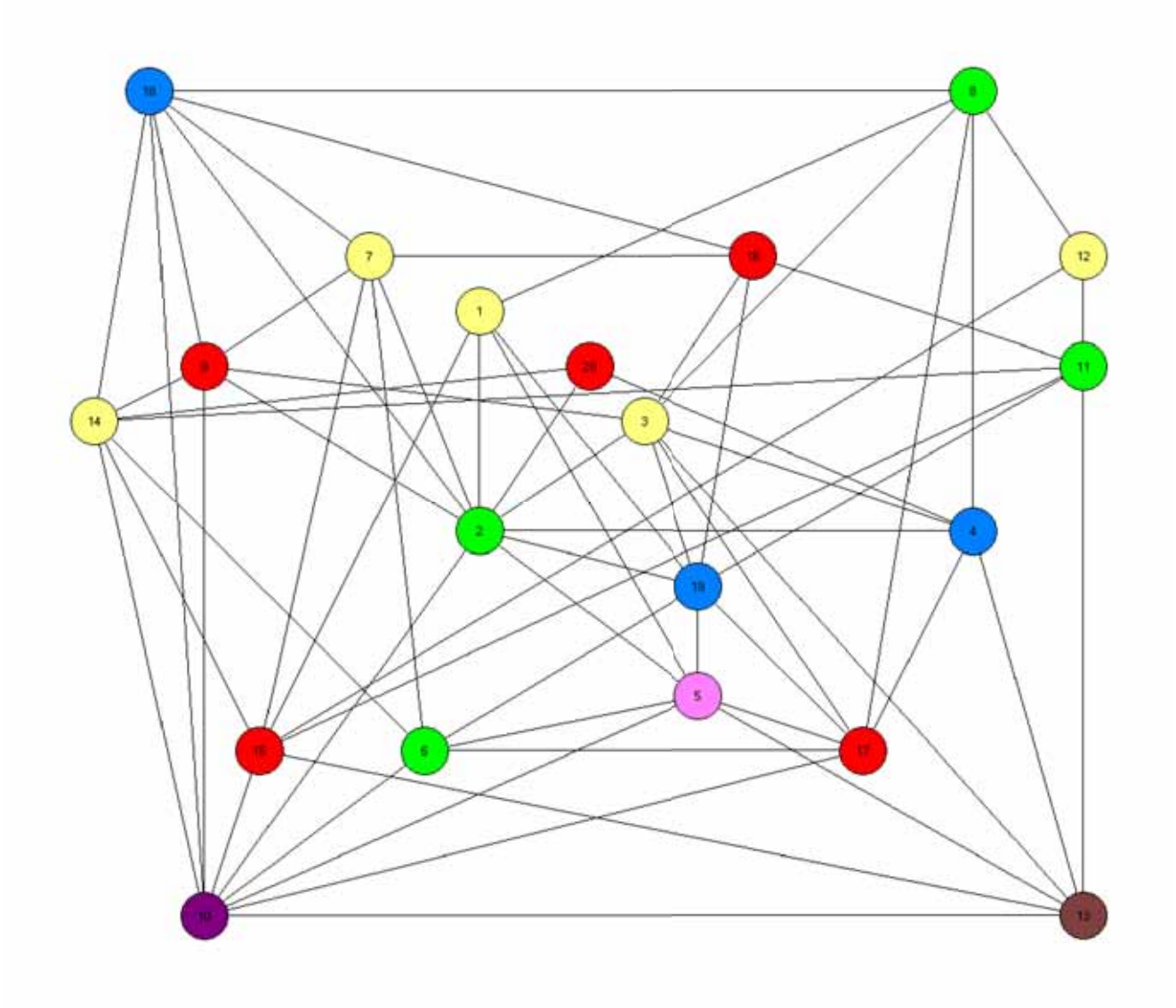

Ilustración 12. Coloración del glotón para el umbral 0.3

En la fase de mejora, se quiere bajar el número cromático difuso a 6 colores, mediante el algoritmo de búsqueda local (Algoritmo 6). Lo primero que se hace es cambiar el vértice 13 por un color aleatorio (ver Ilustración 13), después se calculan las compatibilidades generales (Algoritmo 7), las cuales se ven a continuación

Xn= 6

1 4 1 3 6 4 1 4 2 5 4 1 **3** 1 2 2 2 3 3 2 (coloración propuesta)

Incompatibilidad vértice-vértice
4 -> 13

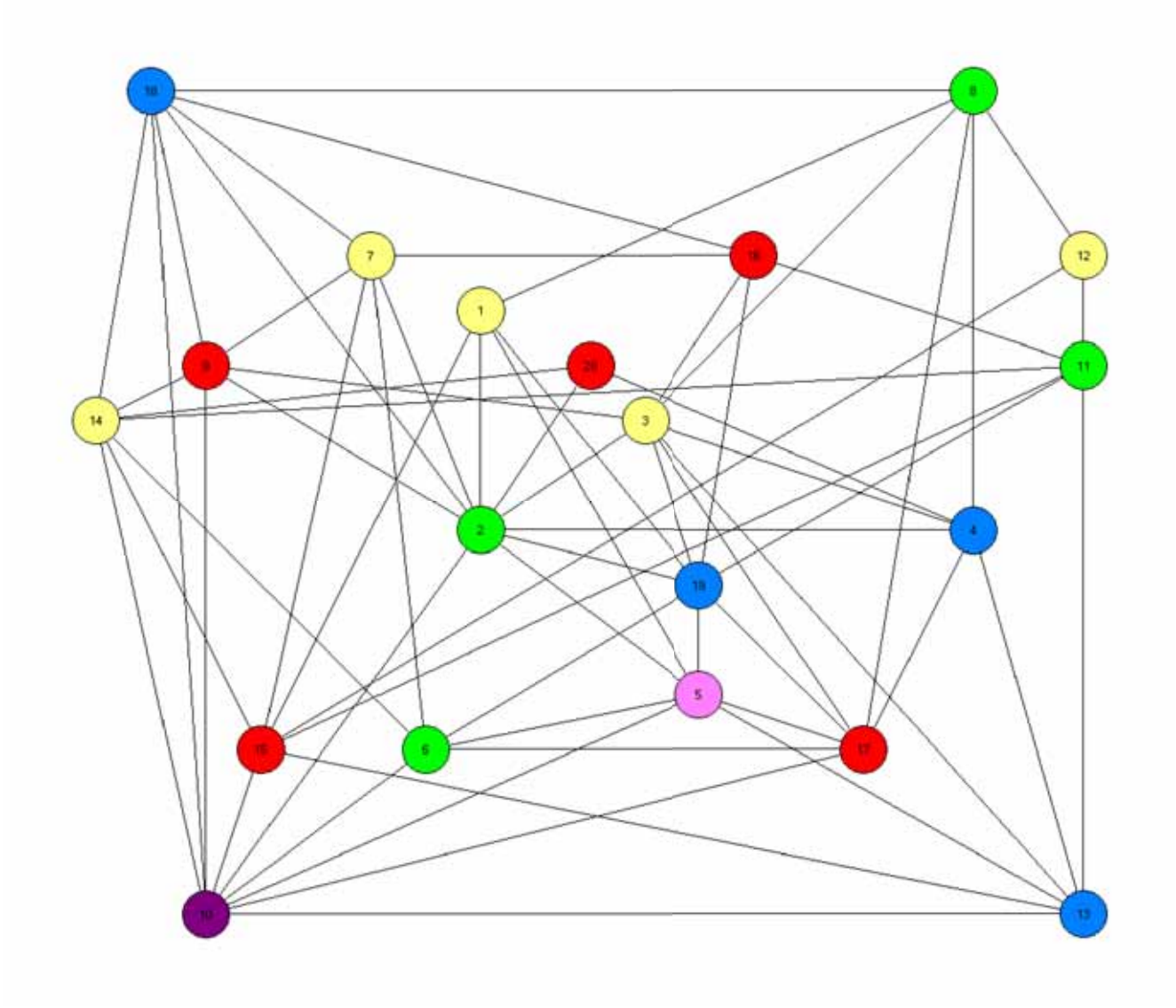

Ilustración 13. Coloración del GRASP con incompatibilidades en la fase de mejora para el umbral 0.3

El algoritmo entra en un ciclo para lograr que se mejore la coloración. Después de varios intentos, el algoritmo logra mejorarla, y actualiza el número cromático difuso y la coloración (ver Ilustración 14) a

X = 6

1 4 1 **6** 6 4 **6** 4 2 5 4 **6** 3 1 2 2 2 3 3 2 (nueva coloración)

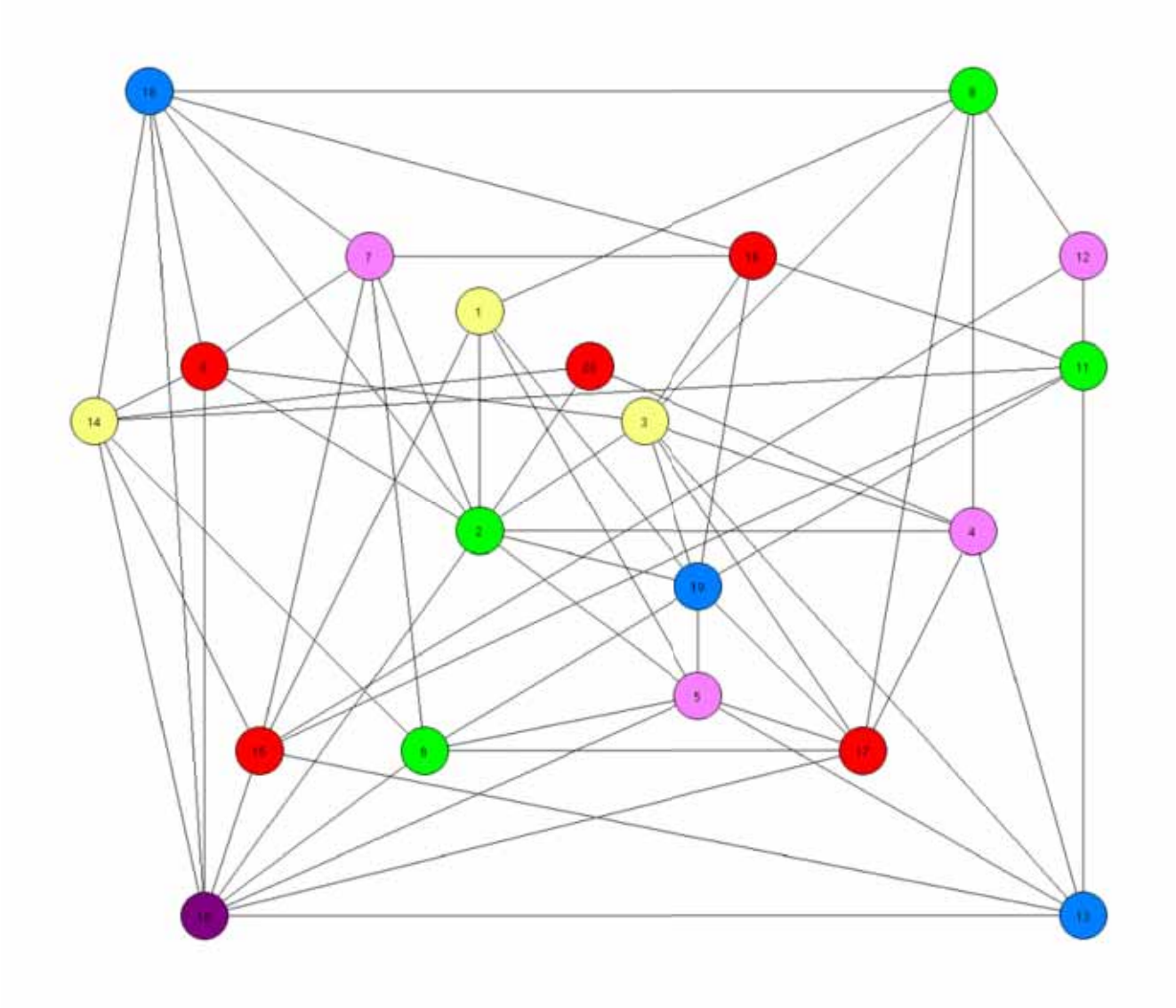

Ilustración 14. Coloración del GRASP en la fase de actualización para el umbral 0.3

Ahora entra en la fase de mejora, pero se quiere disminuir el número cromático difuso a 5. Se cambian los vértices rosas por colores aleatorios (ver Ilustración 15) y observamos las compatibilidades generales que estas ocasionan

Xn= 5

1 4 1 **2** **2** 4 **5** 4 2 5 4 **5** 3 1 2 2 2 3 3 2 (coloración propuesta)

Incompatibilidad vértice-vértice
4 -> 17
4 -> 20
5 -> 17

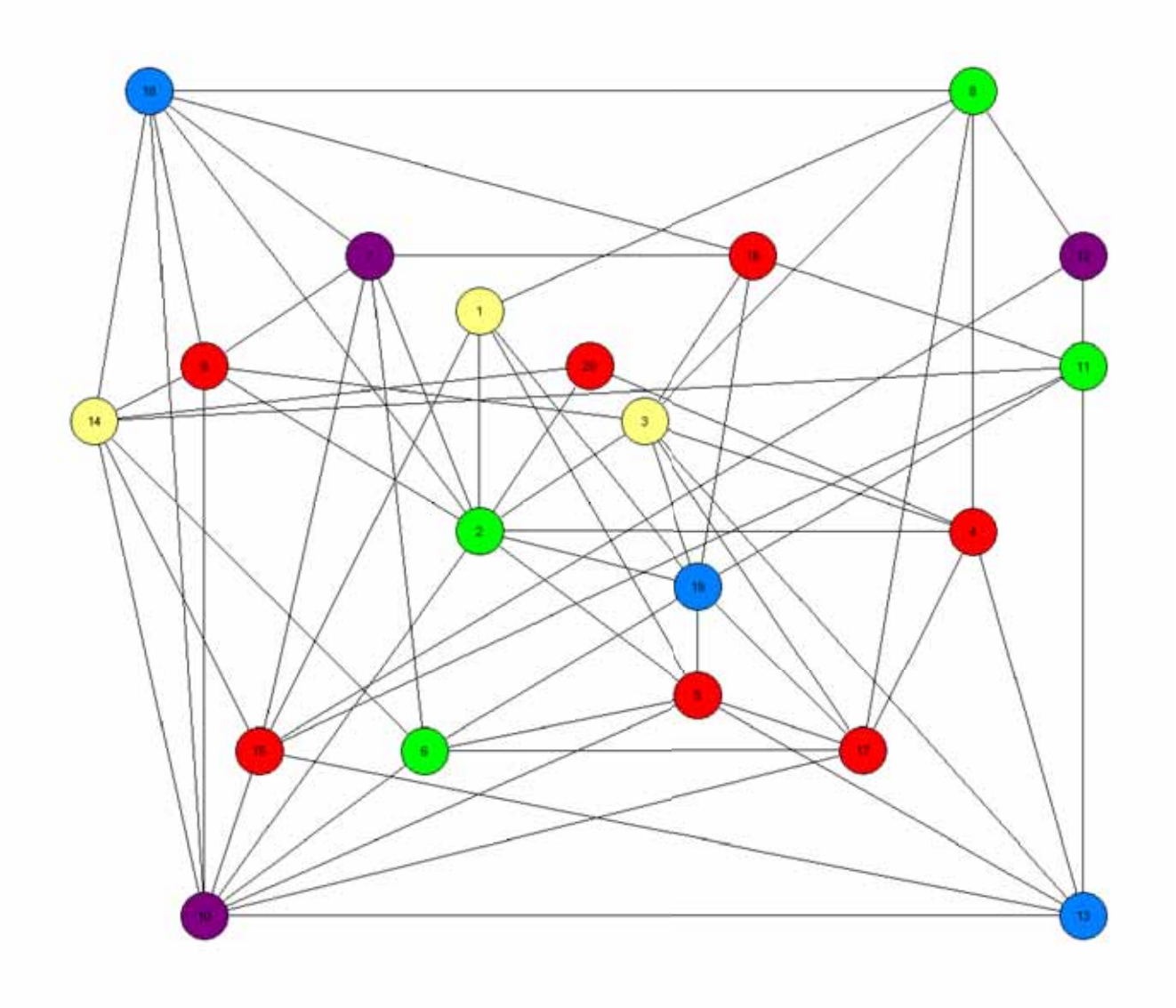

Ilustración 15. Coloración del GRASP con incompatibilidades en la fase de mejora para el umbral 0.3

El algoritmo entra en la fase de mejora. Después de varios intentos, el algoritmo logra mejorarla, y actualiza el número cromático difuso y la coloración (ver Ilustración 14) a

X = 5

1 4 **5** 2 2 4 5 4 2 5 4 **1** 3 1 2 2 **1** 3 3 **3** (nueva coloración)

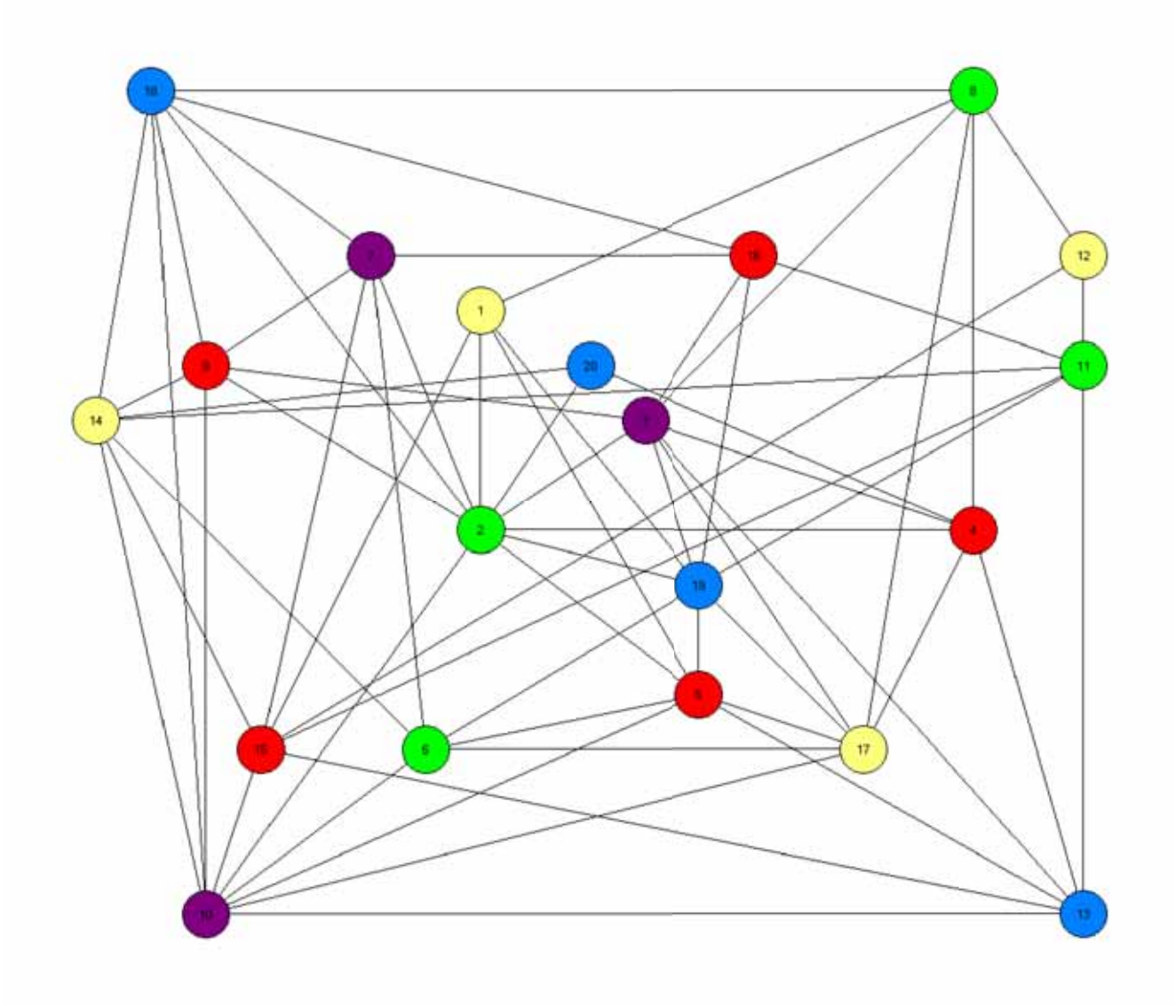

Ilustración 16. Coloración del GRASP en la fase de actualización para el umbral 0.3

Una vez más entra en la fase de mejora, y queremos disminuir el número cromático difuso a 4. Se cambian los vértices morados por colores aleatorios (ver Ilustración 17) y observamos las compatibilidades generales que existen en el grafo.

Xn= 4

1 4 **2** 2 2 4 **1** 4 2 **4** 4 1 3 1 2 2 1 3 3 3 (coloración propuesta)

Incompatibilidad vértice-vértice
2 -> 10
3 -> 4
3 -> 9
3 -> 16
6 -> 10

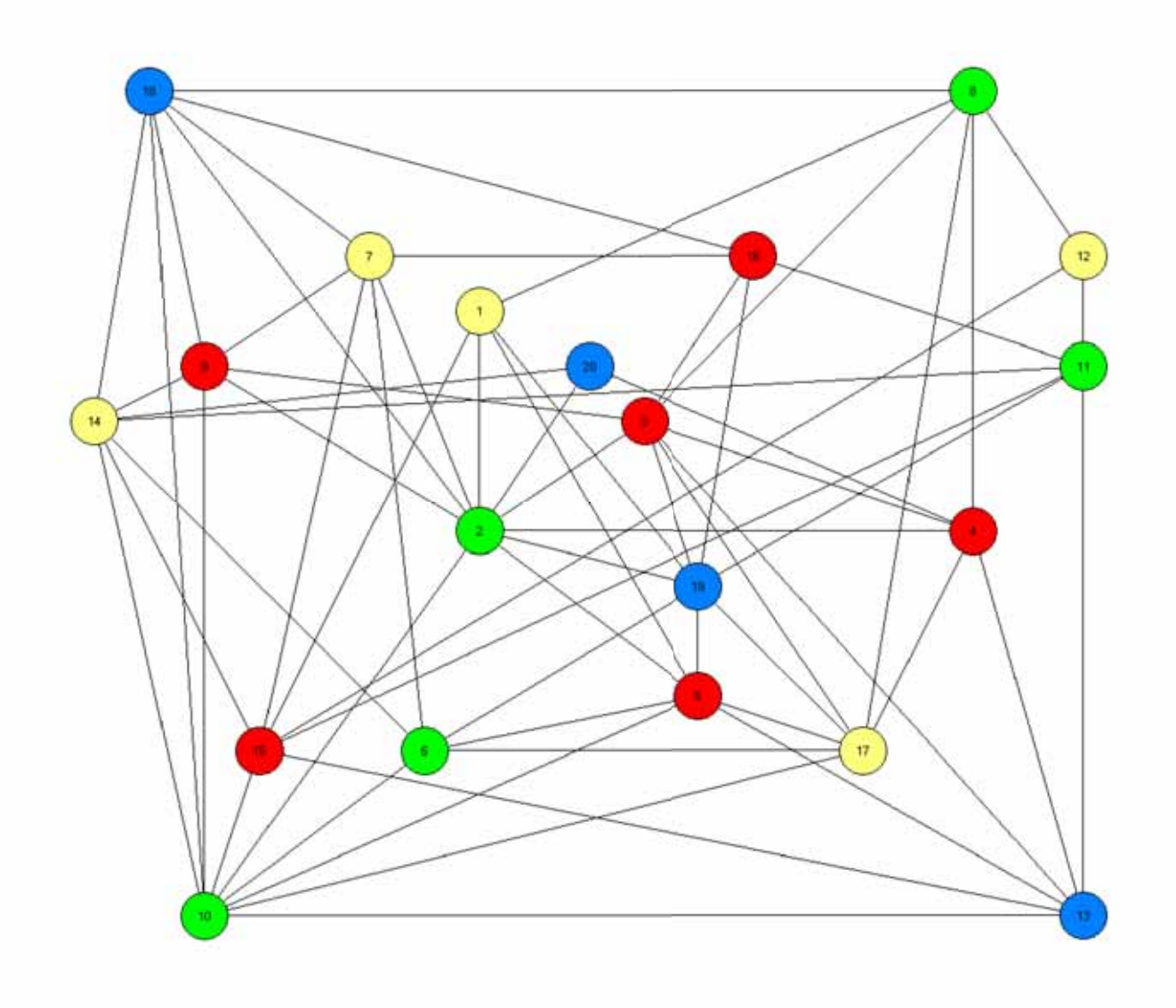

Ilustración 17. Coloración del GRASP con incompatibilidades en la fase de mejora para el umbral 0.3

Después de varios intentos, no se pudo mejorar dicha coloración, se rechazan los cambios y nos quedamos con la coloración anterior válida (ver Ilustración 16).

Resultados

Estos son los primeros resultados obtenidos de la ejecución de un algoritmo GRASP para el problema de Coloración Difusa. Para instancias grandes de 150 vértices, se tienen tiempos del orden de 2 horas 41 minutos, que son comparables con problemas similares (Problema de Coloración Robusta) donde se obtiene un solo valor y una función de distribución.

Cabe señalar que los tiempos obtenidos, son de la ejecución del algoritmo en una máquina con un procesador Intel® Core™ i5-650 (4M Cache, 3.20 GHz) y con 3GB en RAM.

Vértices	Número cromático difuso			Tiempo (s)		Memoria (KB)
	Percentil 0.05	Media	Percentil 0.95	Promedio	Desviación estándar	
20	2	6.85	15	2.3814	0.2473	364
30	3	8.95	20	11.1496	0.4328	372
40	3	10.92	25	35.2793	0.7810	380
50	3	12.57	29	86.8272	0.9660	412
60	3	14.39	33	186.4465	1.7868	428
70	4	16.05	38	352.9486	10.8030	448
80	4	17.75	42	628.1758	16.8849	472
90	4	19.29	45	1036.9365	13.4298	500
100	4	21.01	50	1644.0961	31.6674	528
110	4	22.46	54	2482.9764	45.5122	564
120	5	23.99	57	3635.5795	86.4323	600
130	5	25.55	61	5129.9156	62.9010	640
140	5	27.03	65	7059.9658	117.9130	680
150	5	28.62	69	9690.6214	126.4832	728

Tabla 5. Resumen de resultados de la distribución uniforme

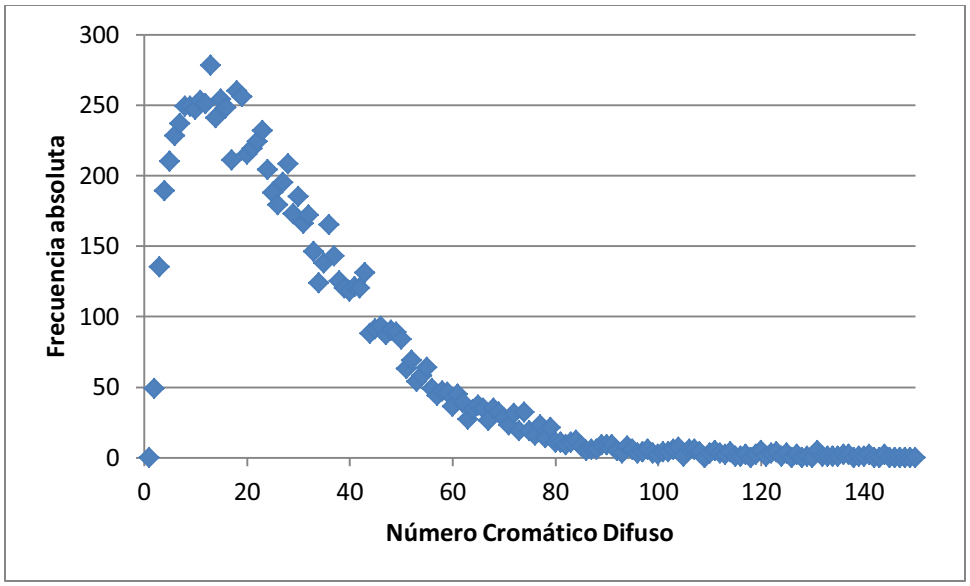

Ilustración 18. Distribución estadística del número cromático difuso del grafo de 150 vértices con distribución uniforme

Cabe notar que en la gráficas de distribución estadística (Ilustración 18 e Ilustración 19), se tiene un sesgo positivo, que puede ser atribuido quizás a que el algoritmo es aproximado.

Vértices	Número cromático difuso			Tiempo (s)		Memoria (KB)
	Percentil 0.05	Media	Percentil 0.95	Promedio	Desviación estándar	
20	1	7.67	19	2.6344	0.2351	364
30	2	10.40	29	11.7767	0.3944	372
40	2	12.57	36	36.2827	0.9628	380
50	2	15.30	45	92.8662	1.8307	412
60	2	17.43	53	196.5306	3.6612	428
70	2	19.55	61	371.3452	6.3035	448
80	2	21.71	68	661.5864	14.2800	472
90	2	23.97	76	1109.8098	11.8341	500
100	2	25.87	84	1718.3147	23.9905	528
110	2	27.87	89	2615.3200	27.0704	564
120	2	30.12	97	3832.0777	58.1001	600
130	2	31.76	101	5410.2248	85.9405	640
140	3	33.85	111	7637.9423	136.2211	680
150	3	33.39	116	10333.5480	174.2515	728

Tabla 6. Resumen de resultados de la distribución triangular

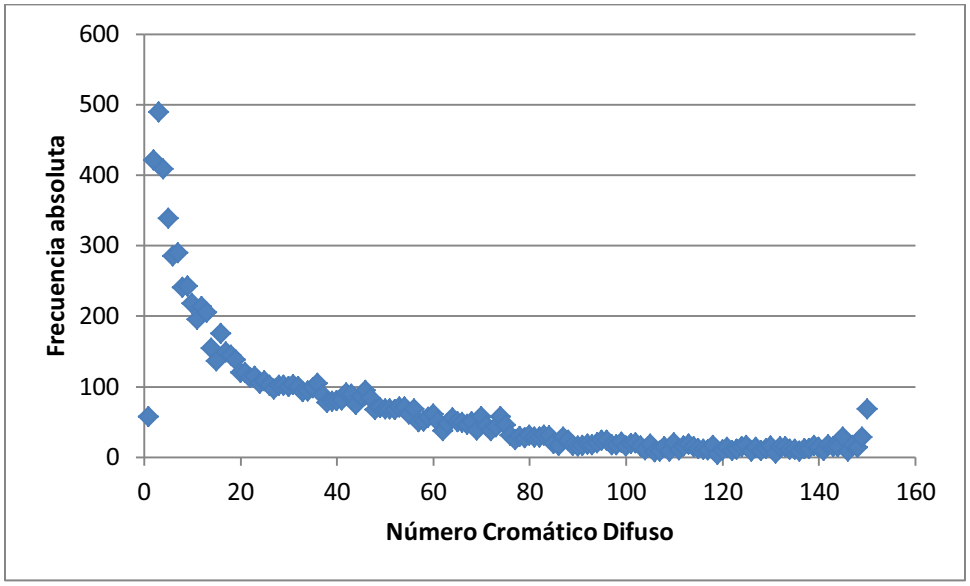

Ilustración 19. Distribución estadística del número cromático difuso del grafo de 150 vértices con distribución triangular

Podemos observar también, en las Tablas 5 y 6, que a medida que el problema crece, el percentil superior (0.95) disminuye respecto al número de vértices en ambos casos, lo que nos muestra que las instancias grandes a medida que aumenta el número de usuarios, se requieren menos recursos para atender a la mayoría de los usuarios, lo mismo se puede decir para el caso promedio.

Notemos que el uso de memoria del algoritmo es pequeño (ver Ilustración 20), ya que utiliza menos de 1 MB. Con 150 vértices utiliza 728 KB, y esto es debido a que sólo necesita crear en memoria lo suficiente para la matriz temporal y un arreglo para la coloración; memoria que es reutilizada para cada umbral.

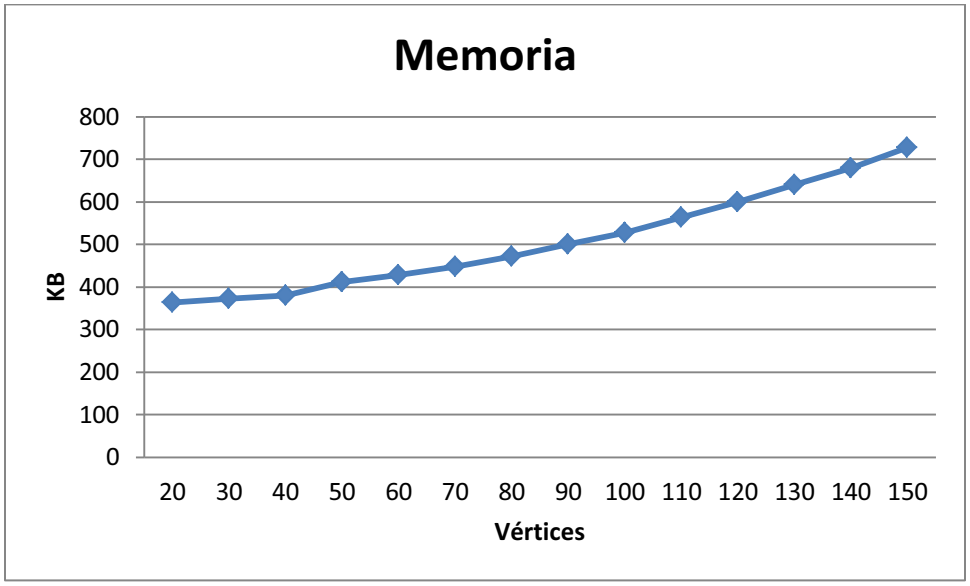

Ilustración 20. Memoria utilizada

Conclusiones

Las matrices simétricas, que usamos tienen distribuciones uniforme y triangular; se llegó a la conclusión de que, dichas distribuciones afectan directamente en las conexiones de pertenencia al umbral en las matrices de adyacencia, por lo tanto las coloraciones generadas son distintas y al graficar la función de densidad se notan muy diferentes. Dichas gráficas se pueden consultar en el Apéndice B.

En el futuro, se pueden introducir otras distribuciones para ver el tipo de función de densidad resultante, ya que aún no se han realizado trabajos sobre esto.

Se puede concluir que nuestra implementación arroja buenos resultados en tiempo y memoria utilizada, para el número de vértices que trabajamos. Si se tiene un número mayor de vértices para resolver algún problema complejo, se tendrá que usar un servidor para que soporte la cantidad de procesamiento, porque en una máquina normal se tardaría mucho.

El realizar muchas iteraciones para cada umbral, es una forma de realizar un muestreo del espacio de soluciones, y puede resultar atractivo pedir que realice muchas, aunque esto aumenta la cantidad de trabajo para el procesador. Basándonos en las observaciones empíricas, se ve que la distribución de la muestra generalmente tiene un valor en promedio inferior al obtenido por un procedimiento determinista, sin embargo, la mejor de las soluciones encontradas supera a la del procedimiento determinista con una alta probabilidad.

Concluimos que la implementación GRASP generalmente es robusta, en el sentido de que es difícil encontrar ejemplos en donde el método funcione erróneamente, dado que tiene límites bien definidos, como el número de intentos para mejorar una coloración.

Algunas de nuestras sugerencias para optimizar el procedimiento son:

- Se puede incrementar el número de umbrales para suavizar la gráfica, si se desarrolla el método utilizando programación con hilos y así aprovechar el tiempo en procesamiento paralelo de un servidor con varios procesadores.
- Se puede mejorar la fase de construcción, si el glotón en vez de generar una solución, dé por lo menos 5 soluciones diferentes, y seleccione de todas ellas la que utilice menos colores, para que al pasarla a la fase de mejora, intente disminuir el número cromático difuso aún más.

Referencias

1. **RAMÍREZ Rodríguez, J.** *Extensiones del problema de coloración de grafos.* Madrid, España : Universidad Complutense de Madrid, 2005. pág. 132. http://eprints.ucm.es/4479/. ISBN 84-669-1855-8.

2. **FEO, T. y RESENDE.** Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization.* 1995. Vol. 2, 1-27.

3. **JOHNSONBAUGH, Richard.** *Matemáticas discretas.* México : Grupo editorial Iberoamérica, 1988. Clasificación QA79.2 J6.48. ISBN 986-7270-46-2.

4. **ABELLANAS M., LODARES D.** *Análisis de algoritmos y teoría de grafos.* México : Macrobit Editores, 1991. Clasificación QA958 A2.5. ISBN 970-604-081-1.

5. **ZADEH, Lotfi A.** Fuzzy Sets, Information and Control. *Department of Electrical Engineering and Electronics Research Laboratory.* [En línea] 1965. [Citado el: 5 de Junio de 2010.] http://www-bisc.cs.berkeley.edu/zadeh/papers/Fuzzy%20Sets-1965.pdf.

6. **CHAIM Sánchez, Sigfrido Carlos.** El Número cromático y la coloración de horarios de exámenes. *Tesis de Licenciatura. División de Ciencias Básicas.* Universidad Autónoma Metropolitana : Azcapotzalco, 1975. Clasificación QA166 CH3.5.

7. **MARTÍ, R.** Procedimientos Metaheurísticos en Optimización Combinatoria. *Universidad de Valencia, España.* [En línea] [Citado el: 2 de Mayo de 2010.] http://www.uv.es/rmarti/paper/docs/heur1.pdf.

8. **VÁZQUEZ Cortés, Alberto A.** Algoritmo Evolutivo para el problema de Coloración Difusa. *Tesis de Licenciatura en Ingeniería en Computación. División de Ciencias Básicas.* Universidad Autónoma Metropolitana : Azcapotzalco, 2009. 29 pp.

9. **FIORITO, Fabián.** La Simulación como una herramienta para el manejo de la incertidumbre. *Universidad del CEMA, Argentina.* [En línea] Mayo de 2006. [Citado el: 10 de Noviembre de 2010.] http://www.ucema.edu.ar/~ffiorito/Handout_Simulacion_y_RISK_06.pdf.

0.0000	0.1896	0.5511	0.6762	0.2113	0.7587	0.5051	0.0585	0.6451	0.7715	0.3547	0.3567	0.6810	0.4601	0.1355	0.7643	0.3346	0.4898	0.2186	0.5716
0.1896	0.0000	0.0263	0.2241	0.1359	0.8118	0.2684	0.7910	0.0106	0.0966	0.7832	0.8522	0.8714	0.7972	0.5842	0.4850	0.6333	0.2279	0.0513	0.0814
0.5511	0.0263	0.0000	0.0252	0.9874	0.5316	0.6707	0.2466	0.1146	0.4967	0.5729	0.6335	0.2152	0.7883	0.7873	0.1156	0.2153	0.9819	0.0250	0.7458
0.6762	0.2241	0.0252	0.0000	0.8982	0.4021	0.7893	0.0830	0.8804	0.8189	0.4115	0.7102	0.1838	0.3568	0.4481	0.6673	0.0592	0.6485	0.3451	0.0833
0.2113	0.1359	0.9874	0.8982	0.0000	0.2184	0.7637	0.7772	0.0830	0.1313	0.9443	0.5655	0.1655	0.5422	0.6781	0.5832	0.0972	0.8647	0.1412	0.5527
0.7587	0.8118	0.5316	0.4021	0.2184	0.0000	0.1613	0.6086	0.5408	0.1418	0.9853	0.4970	0.3898	0.2655	0.9640	0.4286	0.2442	0.9981	0.1455	0.9425
0.5051	0.2684	0.6707	0.7893	0.7637	0.1613	0.0000	0.9349	0.0043	0.5663	0.3406	0.8241	0.8927	0.8129	0.1248	0.1196	0.4002	0.0952	0.9049	0.7757
0.0585	0.7910	0.2466	0.0830	0.7772	0.6086	0.9349	0.0000	0.8125	0.6091	0.4331	0.0142	0.5273	0.4725	0.9227	0.9952	0.1662	0.0146	0.3048	0.6701
0.6451	0.0106	0.1146	0.8804	0.7847	0.5408	0.0043	0.8125	0.0000	0.1418	0.6995	0.5817	0.3464	0.0856	0.4666	0.9321	0.9985	0.0446	0.7674	0.7244
0.7715	0.0966	0.4967	0.8189	0.1313	0.1418	0.5663	0.6091	0.1418	0.0000	0.8518	0.8507	0.0816	0.0944	0.2749	0.5704	0.1924	0.2437	0.5090	0.4411
0.3547	0.7832	0.5729	0.4115	0.9443	0.9853	0.3406	0.4331	0.6995	0.8518	0.0000	0.0031	0.1151	0.2671	0.2000	0.1759	0.6016	0.7325	0.2114	0.3742
0.3567	0.8522	0.6335	0.7102	0.5655	0.4970	0.8241	0.0142	0.5817	0.8507	0.0031	0.0000	0.9438	0.9900	0.2994	0.3898	0.4743	0.4195	0.6921	0.9355
0.6810	0.8714	0.2152	0.1838	0.1655	0.3898	0.8927	0.5273	0.3464	0.0816	0.1151	0.9438	0.0000	0.3630	0.0838	0.9828	0.3861	0.5214	0.6419	0.5155
0.4601	0.7972	0.7883	0.3568	0.5422	0.2655	0.8129	0.4725	0.0856	0.0944	0.2671	0.9900	0.3630	0.0000	0.2471	0.3206	0.8186	0.1006	0.6330	0.2807
0.1355	0.5842	0.7873	0.4481	0.6781	0.9640	0.1248	0.9227	0.4666	0.2749	0.2000	0.2994	0.0838	0.2471	0.0000	0.9624	0.4663	0.5372	0.3865	0.7225
0.7643	0.4850	0.1156	0.6673	0.5832	0.4286	0.1196	0.9952	0.9321	0.5704	0.1759	0.3898	0.9828	0.3206	0.9624	0.0000	0.3532	0.1740	0.1874	0.3758
0.3346	0.6333	0.2153	0.0592	0.0972	0.2442	0.4002	0.1662	0.9985	0.1924	0.6016	0.4743	0.3861	0.8186	0.4663	0.3532	0.0000	0.6574	0.1739	0.3351
0.4898	0.2279	0.9819	0.6485	0.8647	0.9981	0.0952	0.0146	0.0446	0.2437	0.7325	0.4195	0.5214	0.1006	0.5372	0.1740	0.6574	0.0000	0.8361	0.5288
0.2186	0.0513	0.0250	0.3451	0.1412	0.1455	0.9049	0.3048	0.7674	0.5090	0.2114	0.6921	0.6419	0.6330	0.3865	0.1874	0.1739	0.8361	0.0000	0.9138
0.5716	0.0814	0.7458	0.0833	0.5527	0.9425	0.7757	0.6701	0.7244	0.4411	0.3742	0.9355	0.5155	0.2807	0.7225	0.3758	0.3351	0.5288	0.9138	0.0000

Tabla A1. Matriz de una instancia de 20 vértices con distribución uniforme

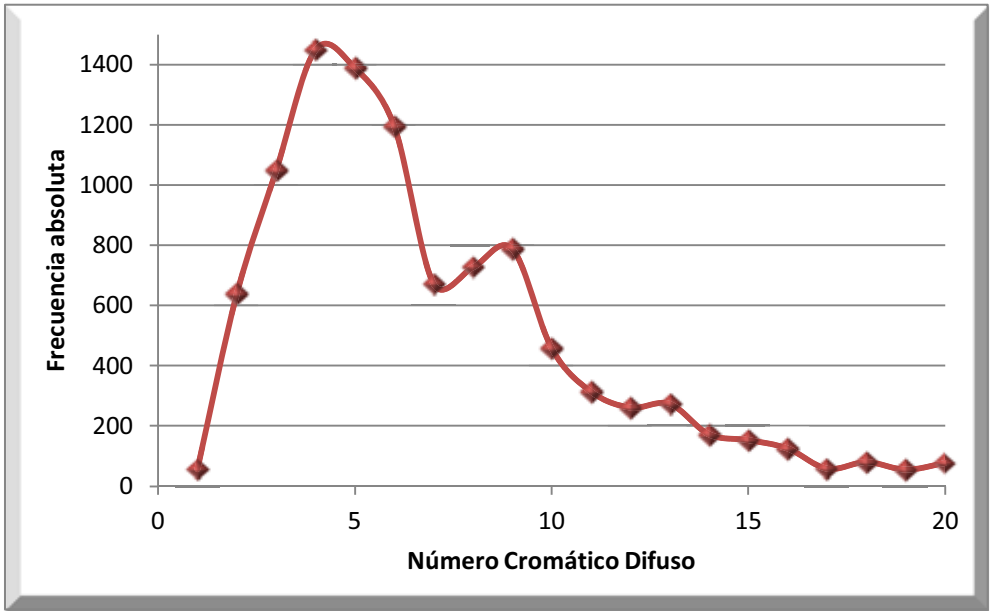

Ilustración B1. Distribución estadística del número cromático difuso de un grafo con 20 vértices y distribución uniforme

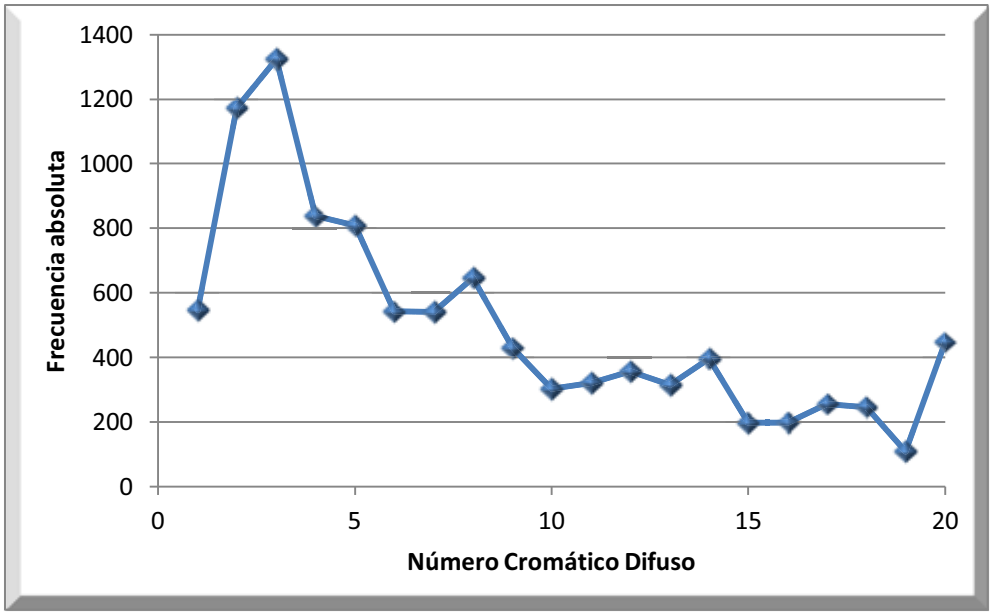

Ilustración B2. Distribución estadística del número cromático difuso de un grafo con 20 vértices y distribución triangular

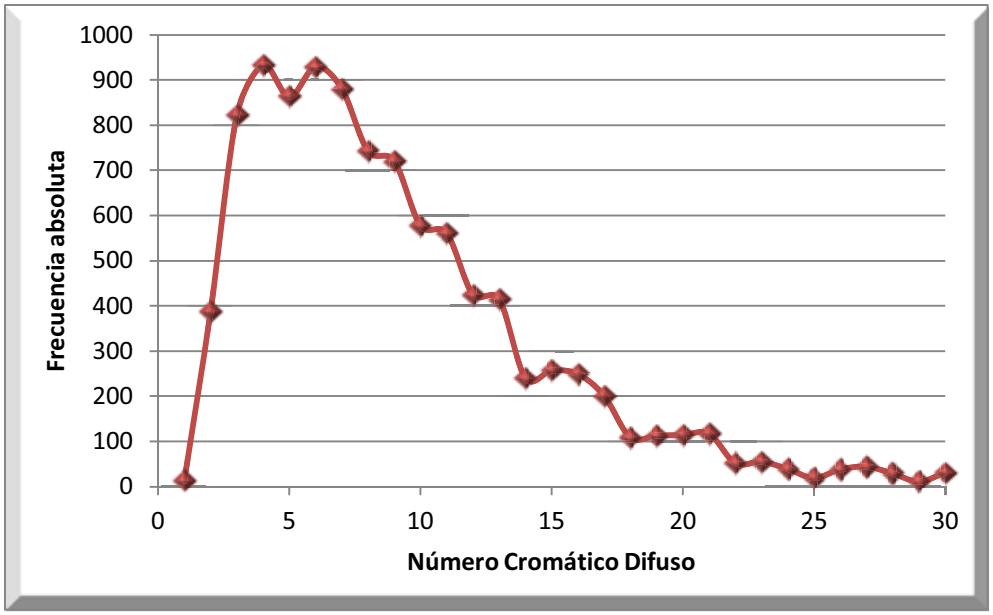

Ilustración B3. Distribución estadística del número cromático difuso de un grafo con 30 vértices y distribución uniforme

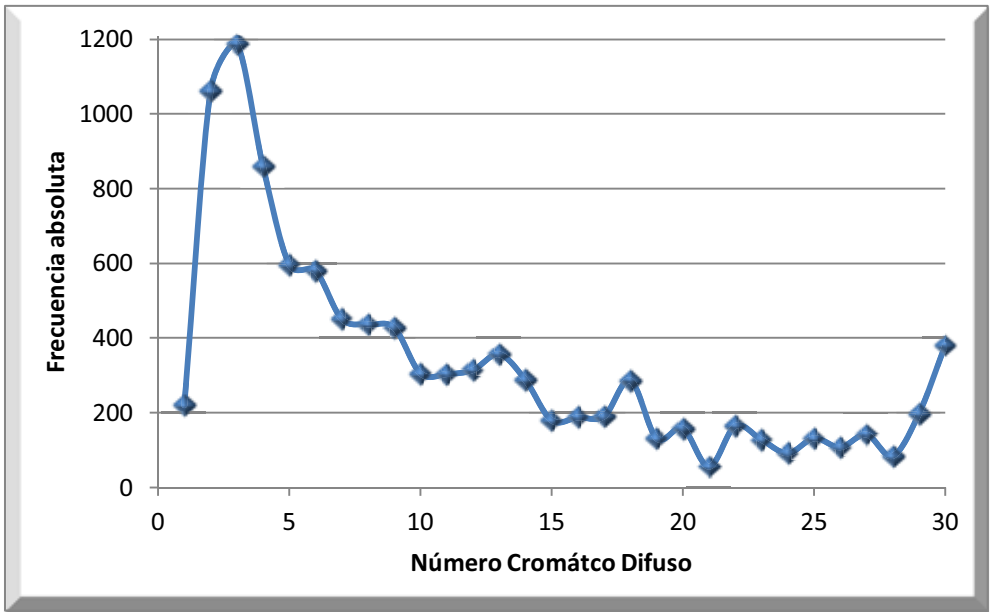

Ilustración B4. Distribución estadística del número cromático difuso de un grafo con 30 vértices y distribución triangular

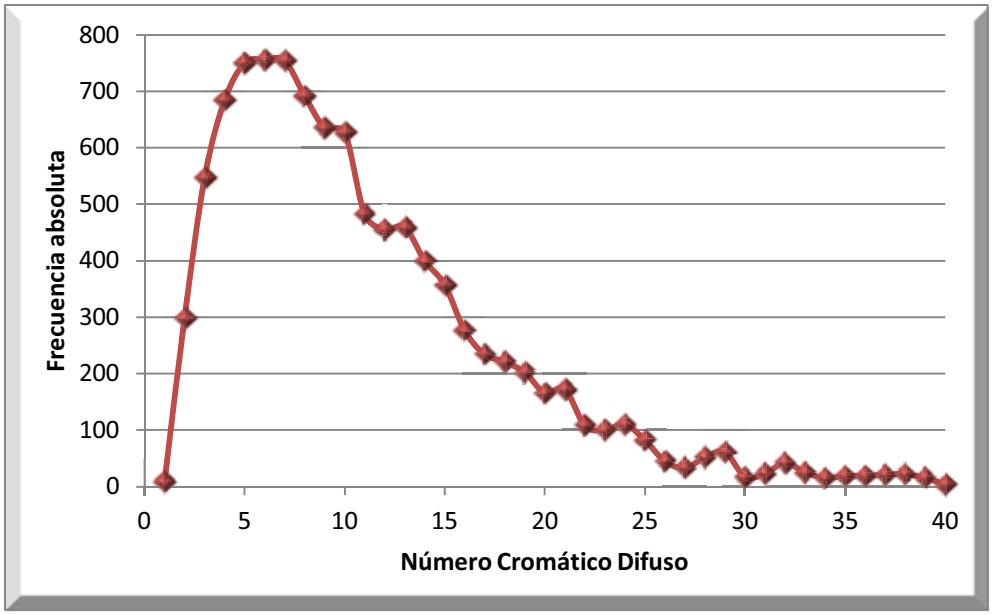

Ilustración B5. Distribución estadística del número cromático difuso de un grafo con 40 vértices y distribución uniforme

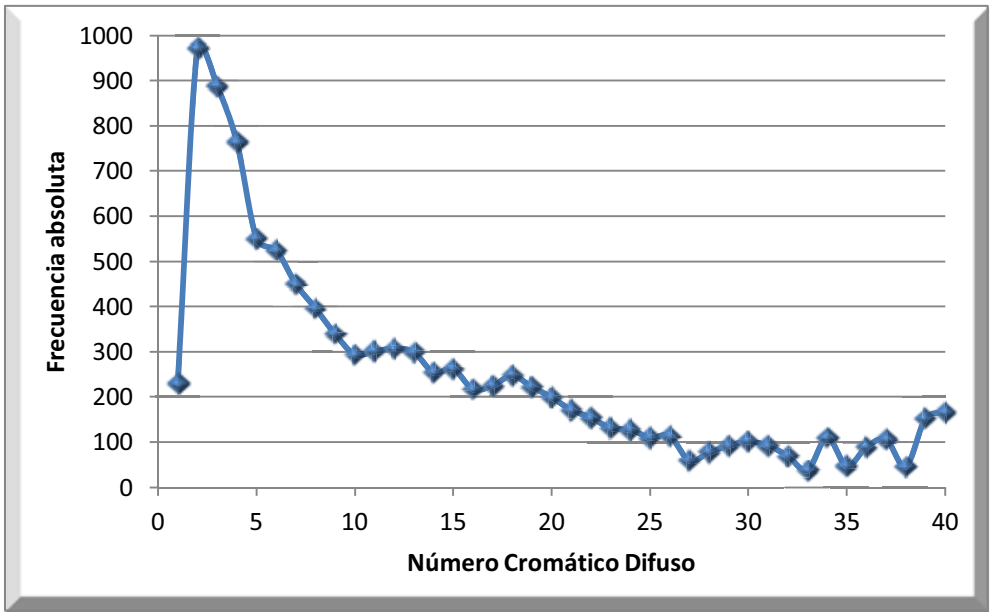

Ilustración B6. Distribución estadística del número cromático difuso de un grafo con 40 vértices y distribución triangular

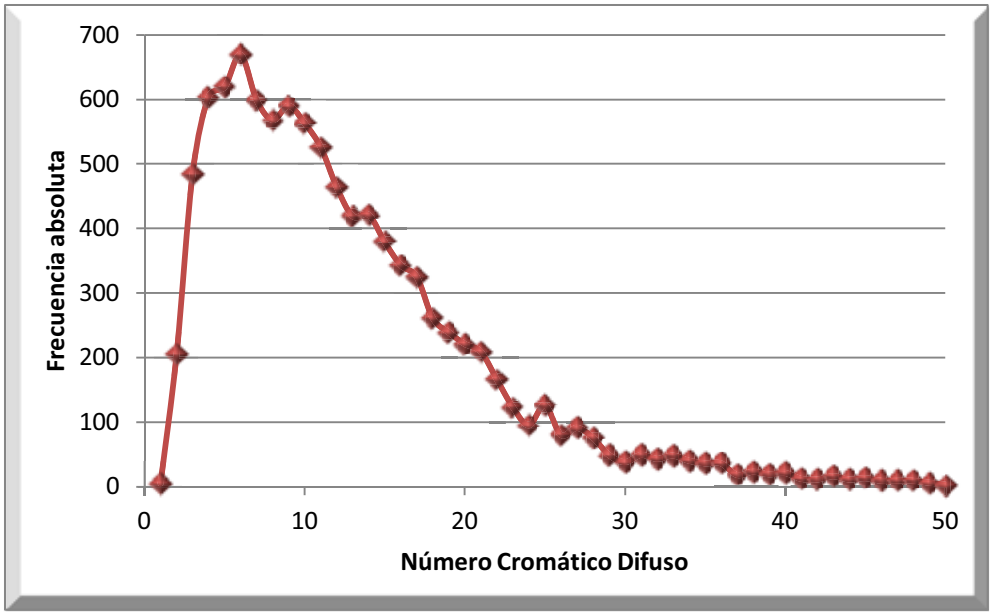

Ilustración B7. Distribución estadística del número cromático difuso de un grafo con 50 vértices y distribución uniforme

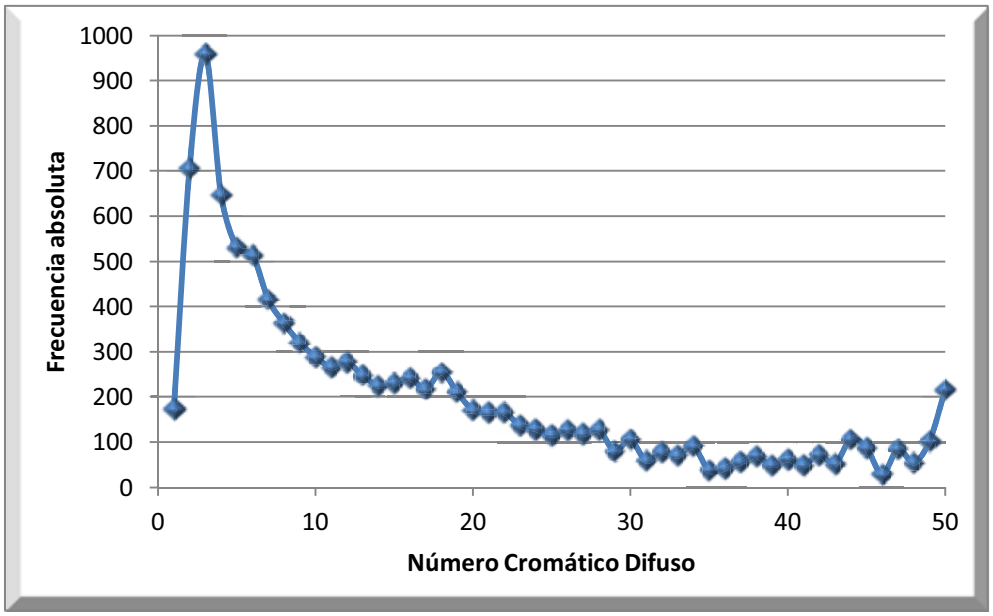

Ilustración B8. Distribución estadística del número cromático difuso de un grafo con 50 vértices y distribución triangular

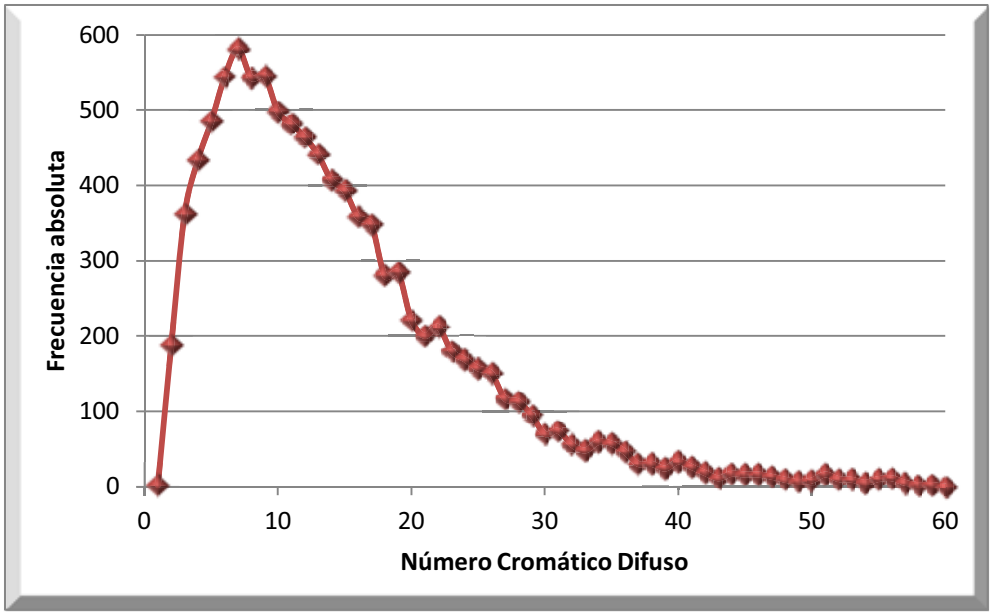

Ilustración B9. Distribución estadística del número cromático difuso de un grafo con 60 vértices y distribución uniforme

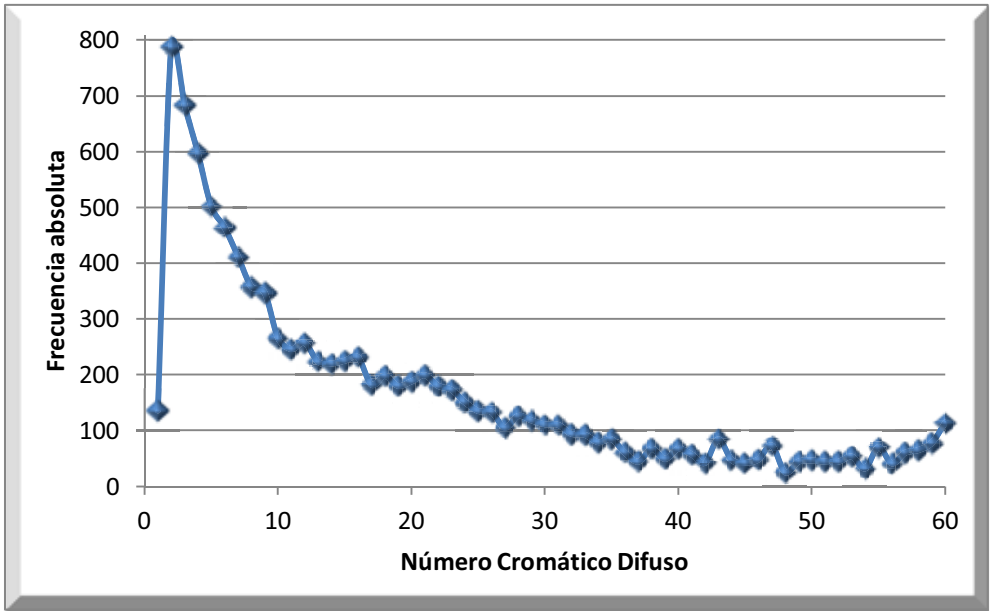

Ilustración B10. Distribución estadística del número cromático difuso de un grafo con 60 vértices y distribución triangular

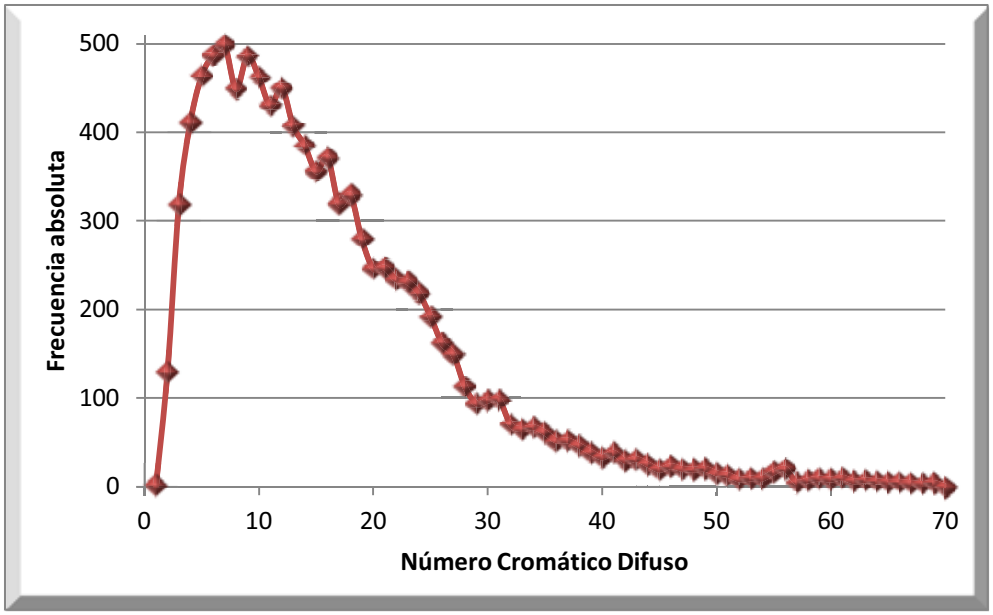

Ilustración B11. Distribución estadística del número cromático difuso de un grafo con 70 vértices y distribución uniforme

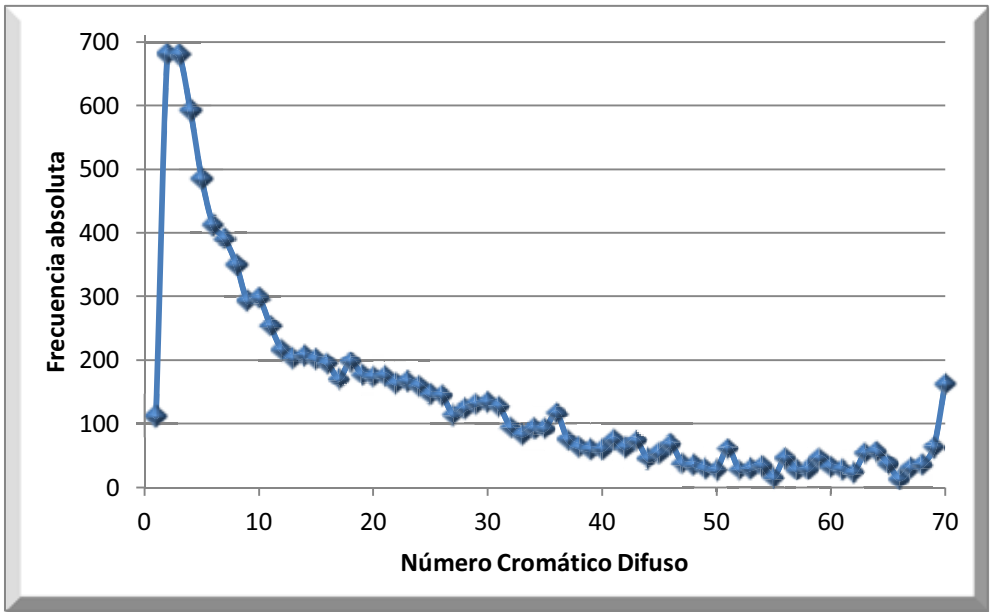

Ilustración B12. Distribución estadística del número cromático difuso de un grafo con 70 vértices y distribución triangular

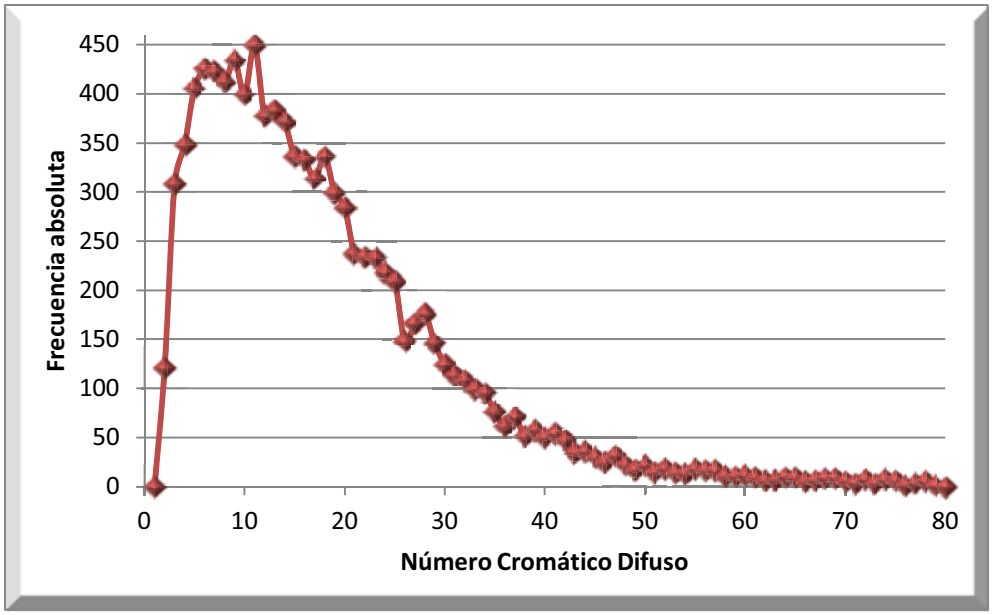

Ilustración B13. Distribución estadística del número cromático difuso de un grafo con 80 vértices y distribución uniforme

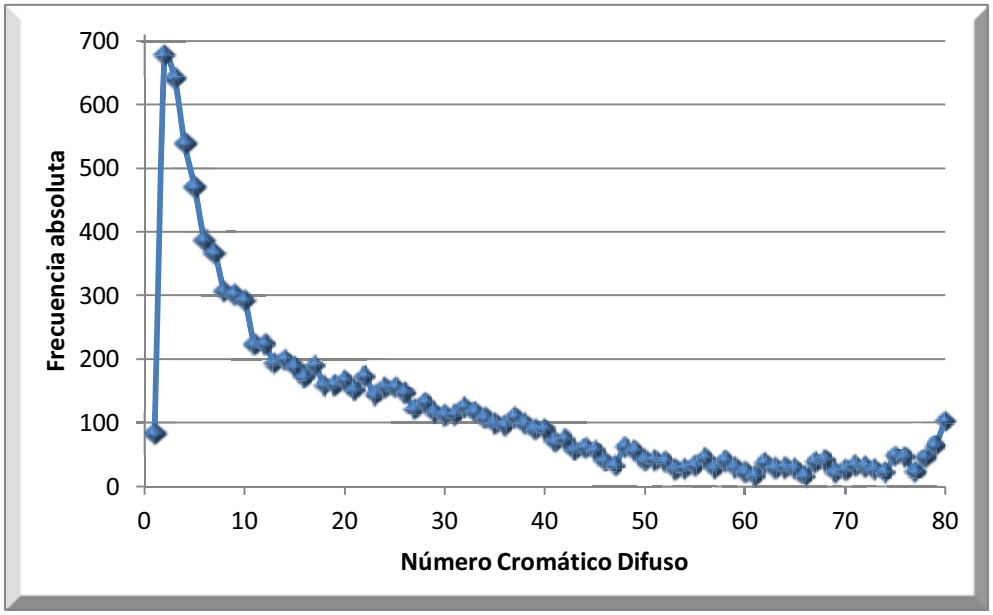

Ilustración B14. Distribución estadística del número cromático difuso de un grafo con 80 vértices y distribución triangular

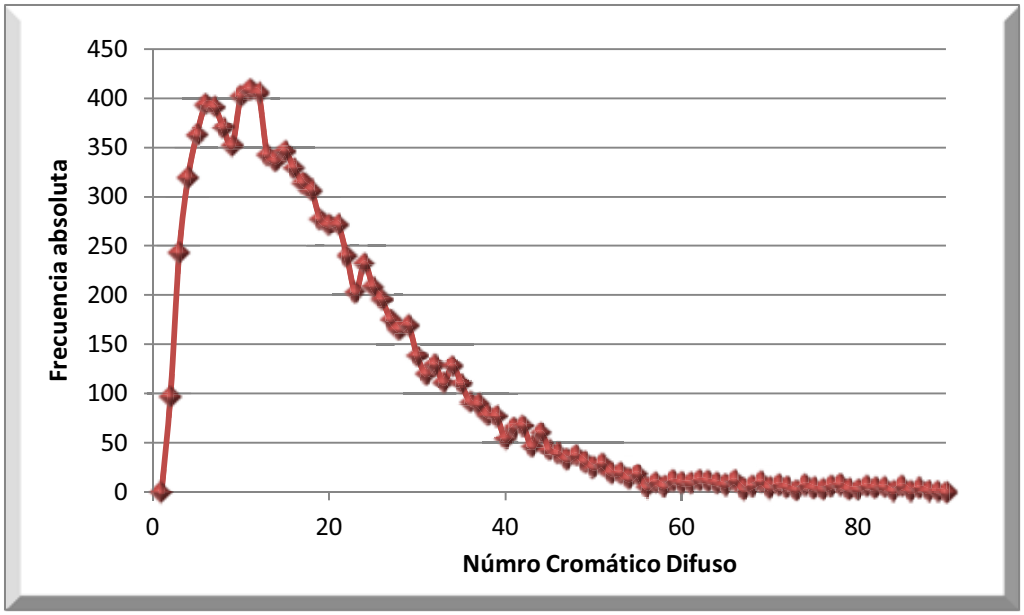

Ilustración B15. Distribución estadística del número cromático difuso de un grafo con 90 vértices y distribución uniforme

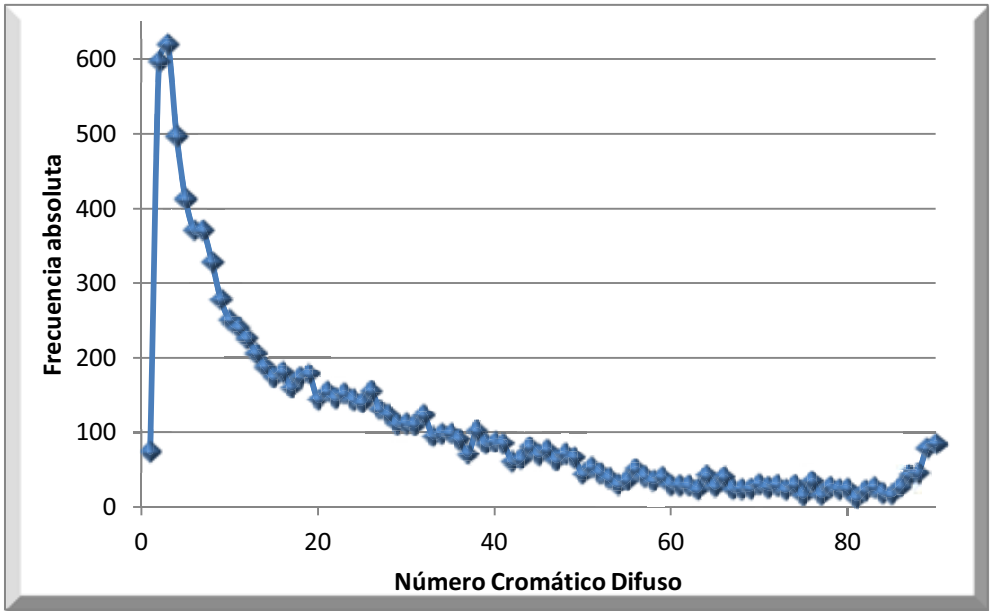

Ilustración B16. Distribución estadística del número cromático difuso de un grafo con 90 vértices y distribución triangular

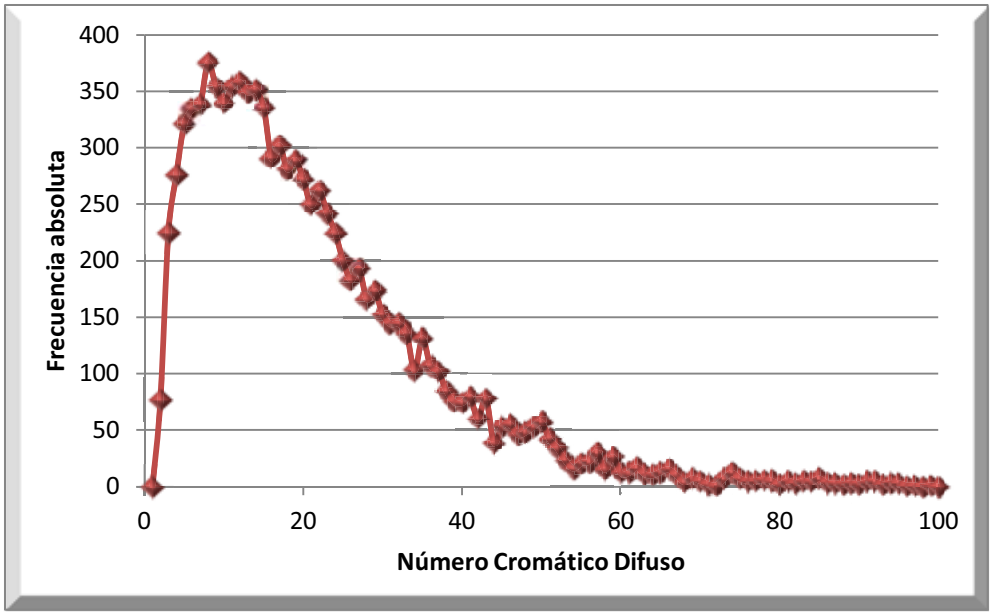

Ilustración B17. Distribución estadística del número cromático difuso de un grafo con 100 vértices y distribución uniforme

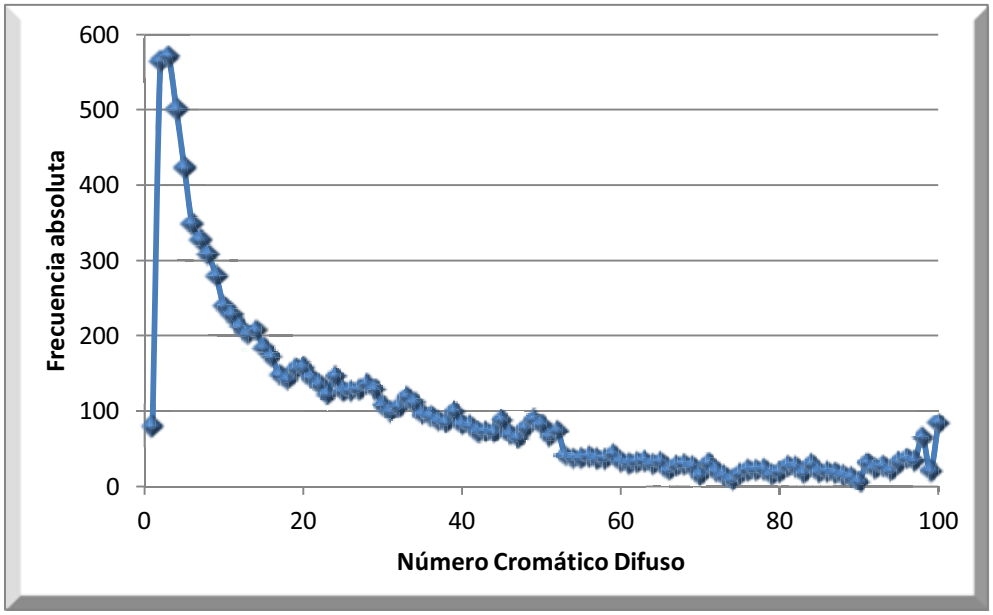

Ilustración B18. Distribución estadística del número cromático difuso de un grafo con 100 vértices y distribución triangular

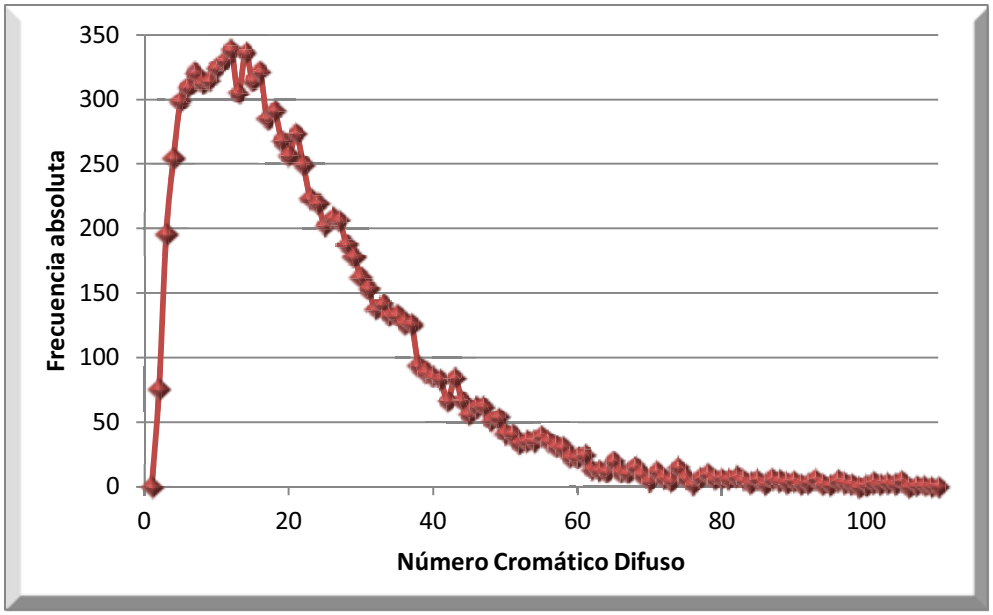

Ilustración B19. Distribución estadística del número cromático difuso de un grafo con 110 vértices y distribución uniforme

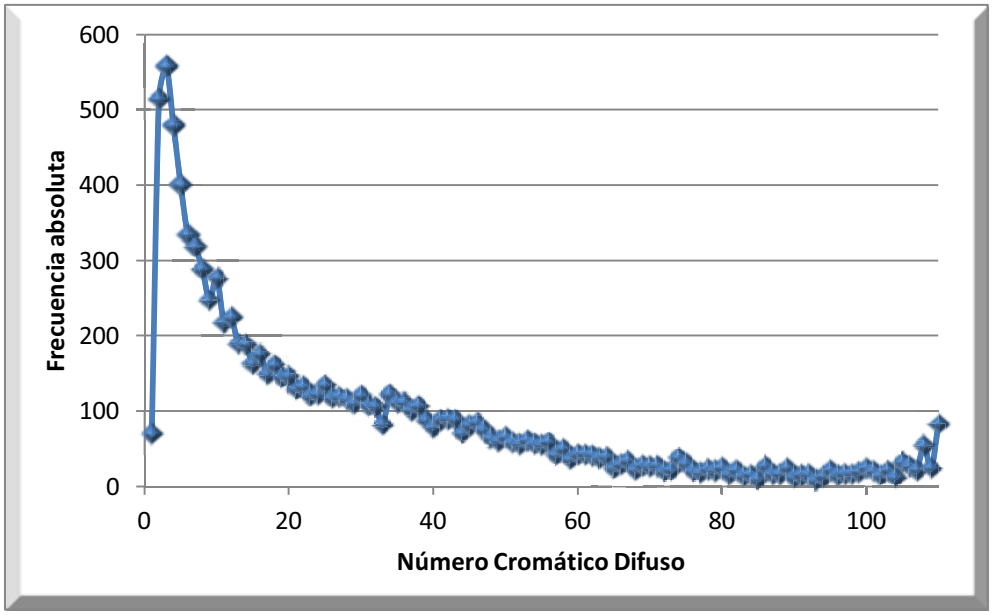

Ilustración B20. Distribución estadística del número cromático difuso de un grafo con 110 vértices y distribución triangular

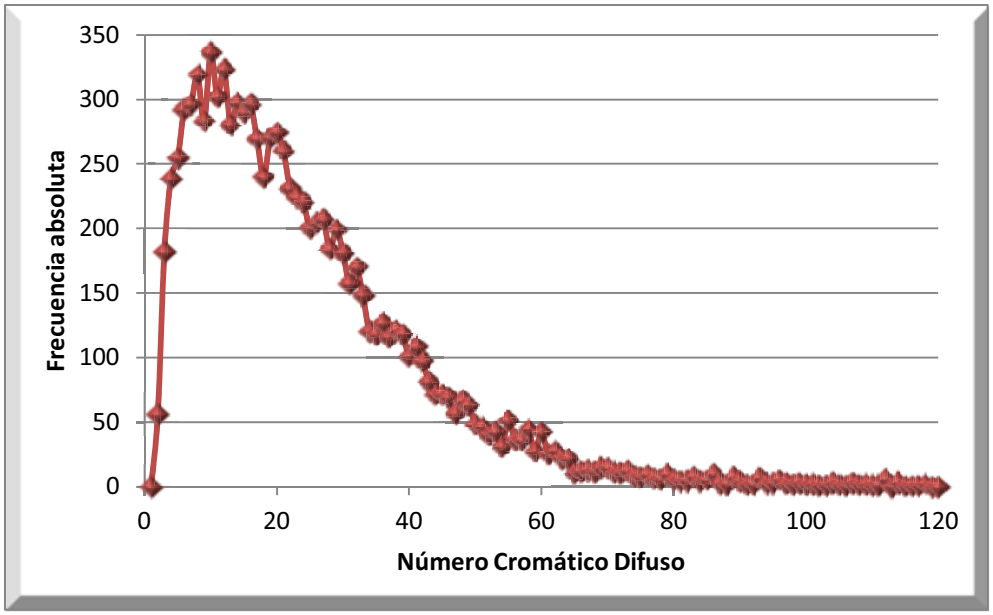

Ilustración B21. Distribución estadística del número cromático difuso de un grafo con 120 vértices y distribución uniforme

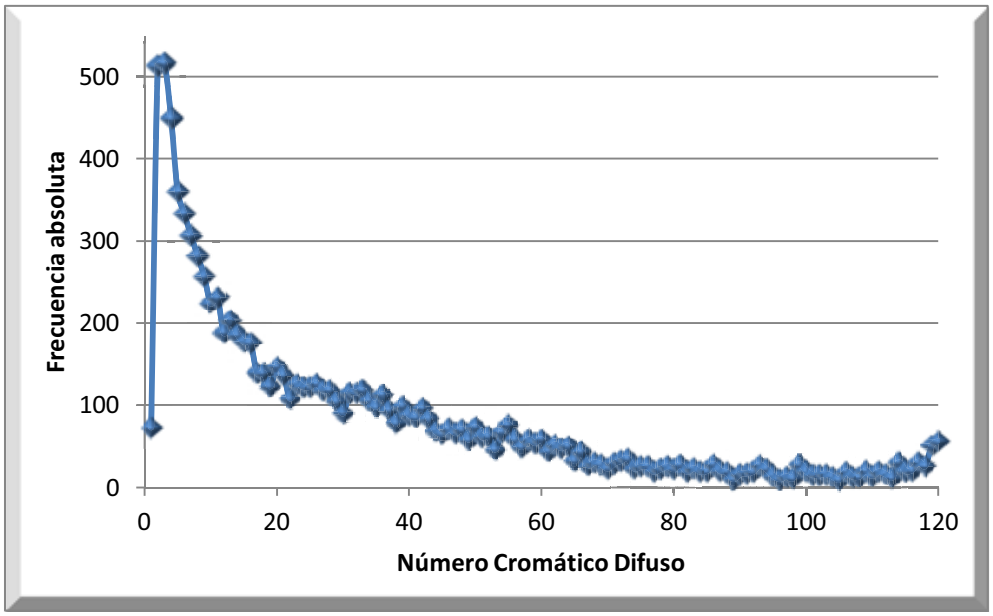

Ilustración B22. Distribución estadística del número cromático difuso de un grafo con 120 vértices y distribución triangular

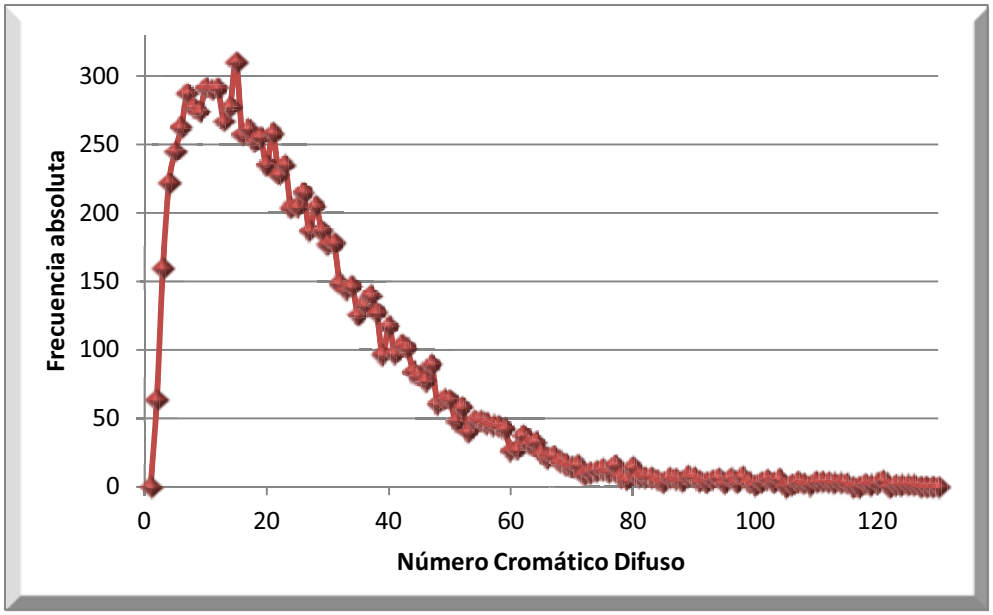

Ilustración B23. Distribución estadística del número cromático difuso de un grafo con 130 vértices y distribución uniforme

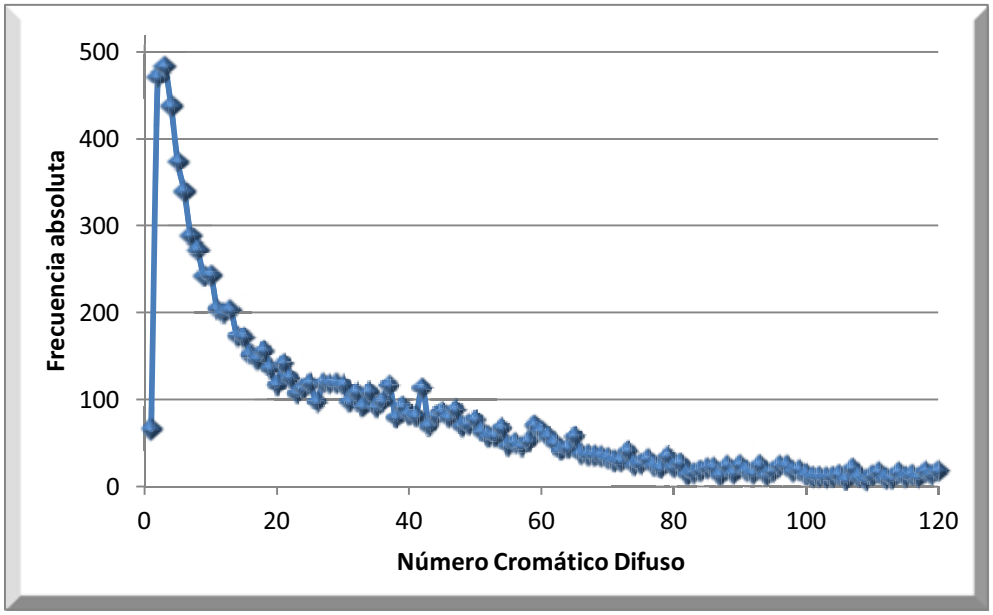

Ilustración B24. Distribución estadística del número cromático difuso de un grafo con 130 vértices y distribución triangular

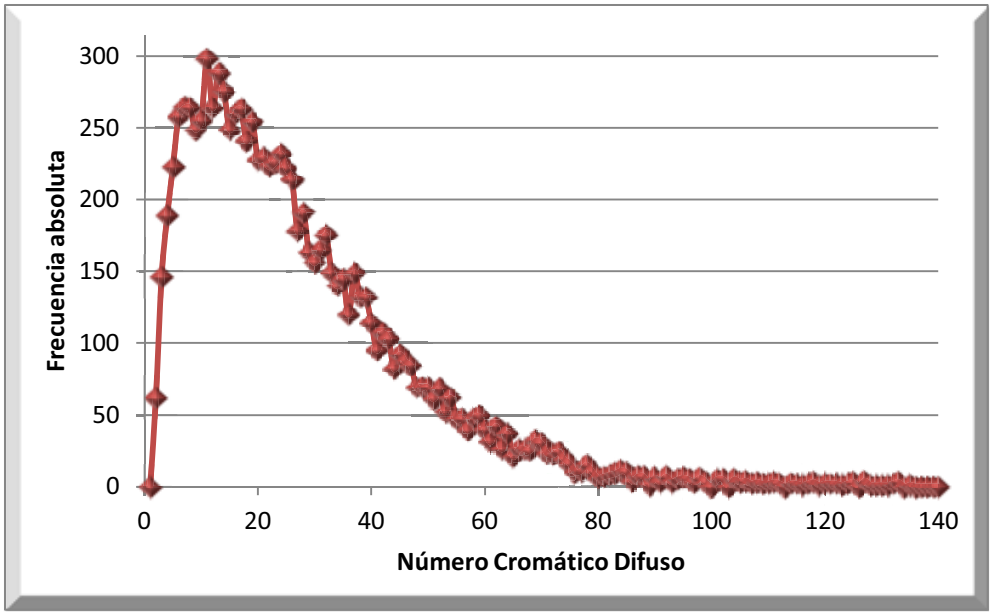

Ilustración B25. Distribución estadística del número cromático difuso de un grafo con 140 vértices y distribución uniforme

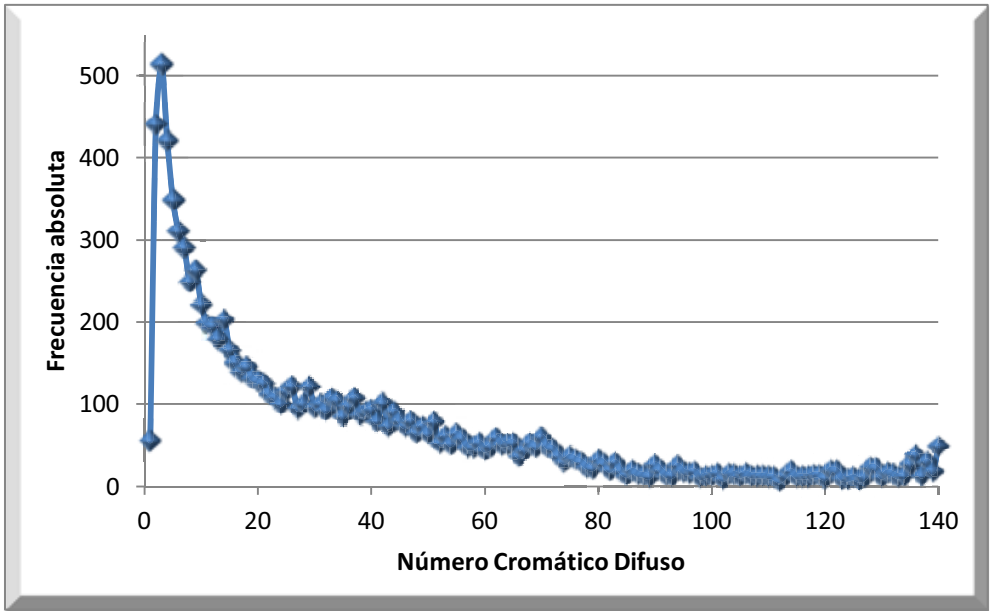

Ilustración B26. Distribución estadística del número cromático difuso de un grafo con 140 vértices y distribución triangular

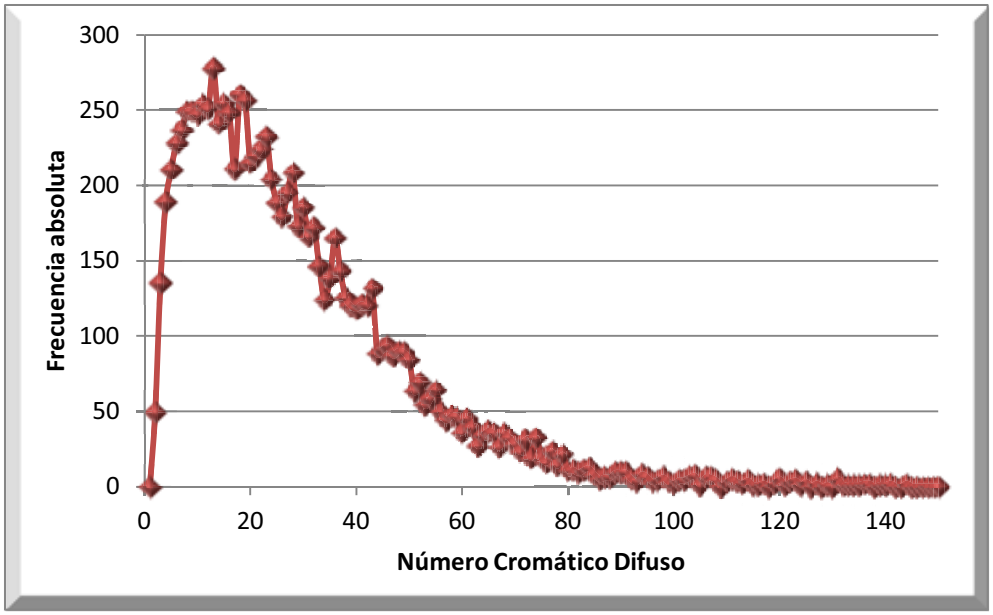

Ilustración B27. Distribución estadística del número cromático difuso de un grafo con 150 vértices y distribución uniforme

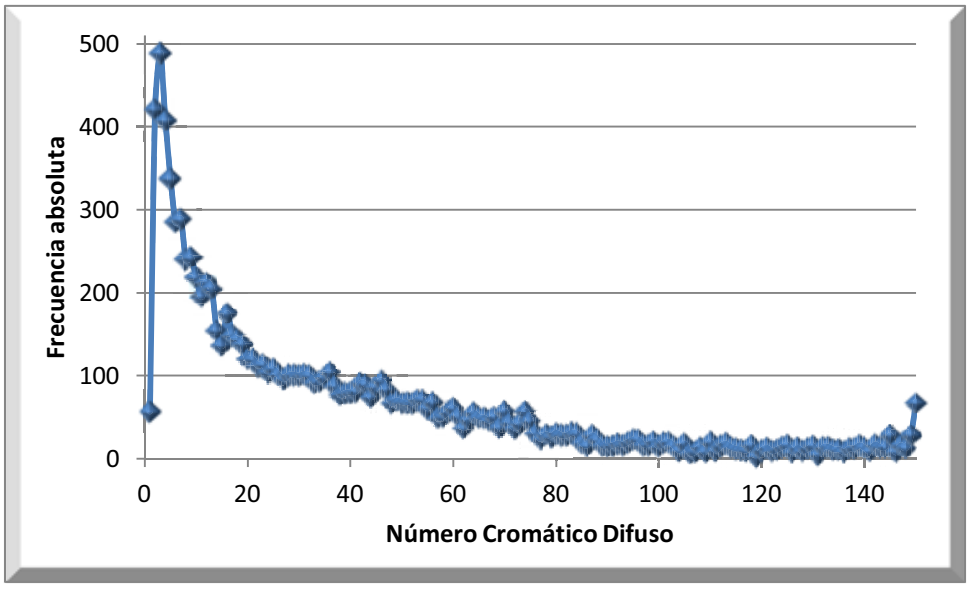

Ilustración B28. Distribución estadística del número cromático difuso de un grafo con 150 vértices y distribución triangular

Apéndice C. Índice de ilustraciones