Ingeniería en Computación

Proyecto Terminal

IMPLEMENTACION DE UN ALGORITMO DE BÚSQUEDA TABÚ PARA RESOLVER EL PROBLEMA DE COLORACIÓN ROBUSTA

Trimestre: 09-P

Agosto de 2009

Iván Frausto Benítez

205201344

Asesor: Dr. Pedro Lara Velázquez

Número económico: 31213

Asesor: Dra. Ma. Lizbeth Gallardo López

Numero económico: 30761

Resumen

Este trabajo presenta una implementación de un algoritmo de búsqueda Tabú como solución al problema de Coloración Robusta. Para determinar la eficacia, eficiencia y tiempo promedio de ejecución del algoritmo implementado, se ejecutó con instancias de prueba conocidas[1]. La originalidad del proyecto radica en que no se ha desarrollado un algoritmo para este problema empleando la técnica de búsqueda tabú.

Introducción

El presente documento proporciona información sobre el desarrollo e **implementación de un algoritmo de búsqueda tabú destinado a resolver el problema de coloración robusta.**

La idea del presenté Proyecto Terminal es proponer un nuevo algoritmo de búsqueda tabú integrando un algoritmo glotón en la solución. El objetivo del algoritmo glotón es buscar soluciones desde un vecindario más estable y por tanto proporcionar una mejor solución al problema de coloración robusta.

Hasta antes de este trabajo, los algoritmos de búsqueda tabú (Tabu search) y glotón (GRASP) no se habían empleado en conjunto. Éste hecho motivó la creación de un nuevo algoritmo, que combine ambas técnicas, con el propósito de resolver el problema de Coloración Robusta. Concretamente del Tabu search se tomaron los conceptos de memoria a corto plazo y memoria a largo plazo; mientras que del GRASP se tomó la técnica de glotón.

Se realizaron pruebas sobre los tres algoritmos: Tabu search, GRASP y Tabú (nombre de nuestro algoritmo). Los resultados muestran que el algoritmo tabú proporciona buenos resultados, con a la incursión del algoritmo glotón, en la mayoría de las instancias de prueba. Con respecto al tiempo, tabú mostró ser más eficaz que el Tabu search.

[1] Matrices que contienen de 20 a 120 vértices que contienen cierta rigidez e incompatibilidad

Búsqueda Tabú

La búsqueda tabú es una metaheurística iterativa que explora un conjunto de soluciones dentro de una cierta vecindad. Ésta surge, en un intento de dotar de inteligencia a los algoritmos de búsqueda local[2]. Según Fred Glover, el primero en definirla: "la búsqueda tabú guía un procedimiento de búsqueda local para explorar el espacio de soluciones más allá del óptimo local" [1].

La búsqueda tabú toma de la Inteligencia Artificial el concepto de memoria, se implementa mediante estructuras simples con el objetivo de dirigir la búsqueda. Este procedimiento trata de extraer información de lo sucedido y actuar en consecuencia. En este sentido puede decirse que hay un cierto aprendizaje y que la búsqueda es inteligente.
La memoria que se implementa en la búsqueda tabú está dividida en dos estructuras: la memoria a corto plazo y la memoria a largo plazo. Ambos tipos de memoria llevan asociados sus propias estrategias y atributos, y actúan en ámbitos diferentes.

La memoria a corto plazo suele almacenar atributos de soluciones recientemente visitadas, su objetivo es explorar a fondo una región dada del espacio de soluciones. La memoria a largo plazo almacena las frecuencias u ocurrencias de atributos en las soluciones visitadas tratando de identificar o diferenciar regiones.

El objetivo de la memoria a largo plazo es diversificar la búsqueda sobre regiones poco exploradas y/o intensificar la búsqueda privilegiando los atributos que se presentan en las mejores soluciones.

La búsqueda tabú es mejor que la búsqueda local ya que permite pasar por soluciones que aunque no sean tan buenas como la actual, nos permiten escapar de óptimos locales y continuar estratégicamente la búsqueda de soluciones aún mejores.

[2] Se define un vecindario para buscar una mejor solución dentro de éste.

Problema de Coloración Robusta

El problema de la 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, esto es, si dos vértices comparten una arista, dichos vértices deben ser pintados con colores diferentes (eliminar incompatibilidad), minimizando así la suma de los pesos (rigidez) de las aristas cuyos vértices comparten el mismo color.

El PCR se utiliza para programar eventos con posibles cambios de último momento, por ejemplo: asignación estable de frecuencias del espectro electromagnético y asignación robusta de registros en microprocesadores.

Trabajos recientes

El problema de coloración robusta es una variante del problema de coloración de gráficas. Esta variante fue planteada en 2003 por Javier Yáñez[3] y por Javier Ramírez[4] en European Journal of Operational Research [2].

Ramírez presentó el PCR como tesis para obtener su grado de doctor en ingeniería. Él mismo propone dos algoritmos para encontrar soluciones aproximadas a la solución del problema de coloración robusta: enumeración parcial, y un híbrido de genético con uno voraz, cuyas soluciones se consideran aceptables después de ser comparadas con las exactas [3].

De igual forma el Dr. Pedro Lara Velásquez, en su tesis para obtener el grado de doctor en Ingeniería presenta dos posibles algoritmos para resolver el problema de coloración robusta: un GRASP y un Algoritmo de Búsqueda Dispersa. Los resultados fueron presentados y comparados con otros algoritmos, y se concluyó, en ese momento, que el primer algoritmo era el más rápido al proporcionar soluciones y el segundo era el que proporcionaba las mejores soluciones [4].

---

3 Departamento de Estadística e Investigación Operativa I, Facultad de Ciencias Matemáticas, Universidad Compútense de Madrid

4 Departamento de Sistemas, Universidad Autónoma Metropolitana Unidad Azcapotzalco

Posteriormente en el artículo titulado “Solución del Problema de Coloración Robusta Utilizando Recocido Simulado” de Gutiérrez Andrade [5], De Los Cobos Silva y Lara Velázquez[2] se destaca que el algoritmo de recocido simulado es el que encuentra actualmente las mejores soluciones con las instancias clásicas del problema [5].

Problema

Implementar un algoritmo de Búsqueda Tabú para resolver el problema de Coloración Robusta.

Diseño de la propuesta

El algoritmo consta de los siguientes módulos principales que son descritos a continuación:

Generador de agentes aleatorios y agentes glotones aleatorizados.

Los algoritmos evolutivos hacen uso de una población para realizar exploraciones de regiones prometedoras. Para realizar dicha exploración se usan dos tipos de soluciones iniciales: agentes aleatorizados y agentes glotones aleatorizados.

Los agentes aleatorios hacen un muestreo aleatorio simple del espacio solución, a partir de los cuales se emigra a regiones prometedoras. La ventaja de este enfoque es que no tiene ningún sesgo en las soluciones de calidad, ya que fueron creadas a partir de puntos al azar del espacio. Por desgracia este enfoque converge con lentitud, además los óptimos locales son de menor calidad que los obtenidos por técnicas que inician en soluciones de calidad.

Los agentes glotones aleatorizados utilizan un algoritmo del mismo tipo para la generación de la población inicial; la ventaja es que las soluciones suelen ser de mayor calidad aunque si no se hace con cuidado, las soluciones pueden estar sesgadas, dejando sin explorar regiones grandes.

---

5 Departamento de Ingeniería Eléctrica, Universidad Autónoma Metropolitana Iztapalapa

Este bloque recibe como entrada una matriz de incompatibilidades/rigidez (nxn, con n entre 20 a 120 vértices). Al termino del proceso entrega como salida una población de agentes iniciales. (nxm, con m, entre 20 y 50 elementos) que se almacena en un arreglo unidimensional calculando la rigidez y la incompatibilidad de la población.

Aplicación de técnica Tabú considerando memoria de corto plazo

La implementación más primitiva de búsqueda tabú se hace realizando búsquedas que consideran sólo una memoria a corto plazo, de esta forma se puede salir de óptimos locales y explorar regiones prometedoras. En este caso el parámetro más importante a considerar es el de memoria (es decir, que tanto recordará el algoritmo), debido a que los valores altos impiden escapar de óptimos locales y valores muy pequeños pueden producir ciclicidad de soluciones así como evitar una convergencia de las soluciones, en este caso se debe escoger cuidadosamente cual es el parámetro adecuado, y si éste depende de algún otro parámetro, como por ejemplo, el tamaño de la instancia.
En este bloque el algoritmo Tabú recibe el agente sin mejorar ejecutándose para generar la memoria a corto plazo de las posibles soluciones.

Creación de memoria a largo plazo a partir del conjunto de soluciones obtenidas

En este bloque genera un estudio estadístico (creación de memoria a largo plazo) de las características que obtienen mejores resultados en alguna instancia en específico. Para hacer un estudio posterior, hay que determinar cual es la característica por estudiar, aunque por lo común tiene una estructura similar a la memoria de corto plazo.
Este modulo, recibe como entrada las posibles soluciones de los agentes generados durante la aplicación de la memoria a corto plazo. Al termino del proceso se tiene como salida una matriz de memoria a largo plazo.

Exploración de nuevas regiones del espacio solución que puedan ser prometedoras.

A partir de la memoria de largo plazo se realizan dos acciones: se estudian las regiones no exploradas para homogenizar el estudio y una vez realizado esto, se generan agentes artificiales (es decir no aleatorios) y de esta manera se hace un estudio a profundidad de las mejores regiones aplicando nuevamente la técnica tabú.
Este modulo recibe como entrada la matriz generada de la memoria a largo plazo. Al termino del proceso se tiene como salida los nuevos agentes propuestos los cuales se almacenan en un arreglo unidimensional. Al final de este punto ya se debe de tener una solución óptima estable.

Presentación de resultados.

Este bloque se imprime el arreglo que contiene buenas soluciones, este resultado será usado en beneficio del usuario. Este modulo tiene como entrada un arreglo de incompatibilidades/rigidez de 20 a 120 vértices. Al termino del proceso tiene como salida un conjunto de soluciones factibles y de alta calidad para el problema propuesto.

Implementación

El programa consta de las funciones principales para su ejecución:

Función main

Función que se encarga de ejecutar las funciones del programa.

Función leer

Función que recibe una matriz nxn de un archivo, cuya información se guarda en una matriz para obtener datos durante la ejecución del programa

Función glotón

Función que recibe un arreglo con valores en cero del mismo tamaño que el número de vértices de la matriz a analizar. A partir del número de colores que es a razón de un 1/3 del número de vértices, se le asigna un color a cada elemento del arreglo. Para llevar a cabo este proceso se tiene un ciclo que recorre todo el arreglo asignando un color de manera secuencial siempre y cuando el valor en el arreglo sea cero y no exista una incompatibilidad. Si existe, el color se asigna aleatoriamente. También se tiene otro ciclo que recorre las posiciones ya asignadas con un color y analiza si hay incompatibilidad. Si no la hay, reutiliza un color ya colocado en una posición con el fin de ocupar el mínimo de colores. Al terminar la asignación de colores se tiene un conjunto de agentes propuestos con una incompatibilidad de valor cero y cierta rigidez.

Función objetivo

Esta función recibe un arreglo de agentes propuestos generado en distintas etapas de la ejecución del algoritmo. Tiene dos ciclos que se utilizan para comparar un elemento del arreglo con respecto a otro elemento del mismo. Si al comparar son iguales, se busca en la matriz leída que valor se tiene en esa posición para obtener la rigidez y la incompatibilidad sumando el valor de estos en una variable.

Función imprime

Función que recibe el arreglo con los agentes propuestos del glotón, memoria a corto plazo y largo plazo generados durante la ejecución del programa. A través de un ciclo de n número de vértices imprime en pantalla el contenido del arreglo así como los valores de incompatibilidad y rigidez.

Función Memoria_Corto

Esta función recibe el arreglo generado por el glotón. Se ejecuta 100 veces con la finalidad de encontrar una menor rigidez para nuestro arreglo solución. Para efectuar dicha ejecución depende de tres funciones que se describen a continuación:

- **Función Mejora_Rig.** Esta función recibe el arreglo a evaluar, un valor de tipo entero que se elige aleatoriamente y la posición que se obtiene durante el recorrido del arreglo. Esta posición es buscada en un arreglo para saber si se encuentra dentro del arreglo o no. Si lo esta, el valor de la posición se le suma un mas uno y se vuelve a analizar el nuevo valor para determinar si esta dentro del arreglo o no. En caso contrario, se obtiene el valor de esa posición en el arreglo guardándolo en una variable temporal. Se utiliza el valor aleatorio agregándolo en la posición del valor que guardamos anteriormente y calcula con la función objetivo si se puede obtener una menor rigidez. Si es así, el valor aleatorio se guarda en un arreglo de la función memoria. En caso contrario se regresa el valor original que guardamos anteriormente en una variable a la posición que evaluamos.
- **Función Buscar_Tabu.** Esta función recibe un valor el cual busca en un arreglo, si lo encuentra regresa ese valor para indicar que es tabú sino regresa un -1.
- **Función Memoria.** Esta función recibe un valor que es almacenado en una cola de 5 elementos. Se utiliza este número de elementos ya que se tienen mejores resultados. Al guardar el valor, no se utiliza hasta que sale de la cola, siendo por regla que el primero que entra es el primero que sale. Con esto se logra que el valor no se utilice, prohíba o sea tabú por lo menos en 5 búsquedas y así ampliar la exploración de posibles soluciones.

Función Memoria_Largo

En esta función crea un archivo que contiene una matriz (nxn) (donde n es el número de vértices), los datos que representan los cambios que la memoria a corto plazo encuentra a una mejora de la solución. A partir de esta matriz se obtendrá un nuevo arreglo de agentes propuestos para encontrar una mejor solución. Esta función depende de las siguientes funciones para realizar su tarea.

- **Función Cambios.** Esta función recibe el arreglo generado por la función *Memoria_Corto*. Su tarea es comparar cada valor del arreglo con el resto. Si son iguales, las posiciones de esos valores se representan bidimensionamlemente como una posición para llenar la matriz contenida en el archivo sumando lo que se encuentre en esa posición con más uno.

- **Función Nuevo_Gloton.** Recibe un arreglo con valores en cero el cual se va llenando deacuerdo a la información que nos proporcione la matriz contenida en el archivo. Esta información nos permite obtener un nuevo arreglo de agentes propuestos. Para realizar la tarea coloca en la primera posición del arreglo un color, Estos parámetros son utilizados en la siguiente función que se describe continuación:

    - **Función Tendencia.** Recibe como parámetros la matriz contenida en el archivo, un valor que representa una posición en el arreglo de los nuevos agentes propuestos y un valor que representa un color. Su tarea es recorrer cada fila de la matriz buscando el valor máximo de cada una de ellas. Al encontrar el valor máximo, vuelve a recorrer la misma fila teniendo en cuenta las siguientes condiciones. Si el valor máximo dividido con uno punto cinco (ya que mejora la solución) es menor al valor que se tiene al recorrer la fila de la matriz y además el valor en el nuevo arreglo de agentes propuestos es cero, se coloca el color que se recibió como parámetro.

Esta función se repite tantas veces como sea el número de vértices. Al terminar se tendrá un nuevo arreglo de agentes propuestos que entrara nuevamente a la función *Memoria_Corto* para obtener un arreglo de agentes propuestos óptimo.

Función explorar

Esta función recibe los arreglos de los agentes propuestos generados tanto de la función *gloton y* función *Memoria_Largo.* Estos arreglos son evaluados en la función *Memoria_Corto,* para obtener un arreglo de agentes propuestos de solución óptima. Para llegar a esta solución se tiene un ciclo que se repite 10 veces (ciclos suficientes para encontrar una buena solución en un tiempo razonable) en cada repetición el arreglo a evaluar entra a la función *Memoria_Corto*, obteniendo su análisis a través de la función *objetivo.* Si la incompatibilidad es diferente cero se repite el ciclo. De lo contrario se compara el valor de la rigidez obtenida con el valor de la mejor menor rigidez que se tiene hasta el momento almacenado en una variable temporal. Si es menor al valor almacenado se guarda ese valor en la variable temporal y el arreglo que contiene a los agentes propuestos se guarda en un arreglo temporal. Inmediatamente después la función de *Memoria_Largo* hace su tarea. En caso contrario no hay cambios en las variables temporales. Al terminar el ciclo imprime el arreglo temporal que contiene a los agentes propuestos óptimos, así como su rigidez e incompatibilidad calculados con la función *objetivo*.

A continuación se muestra un diagrama de flujo de cómo se ejecuta el programa así como el pseudocódigo de cada función.

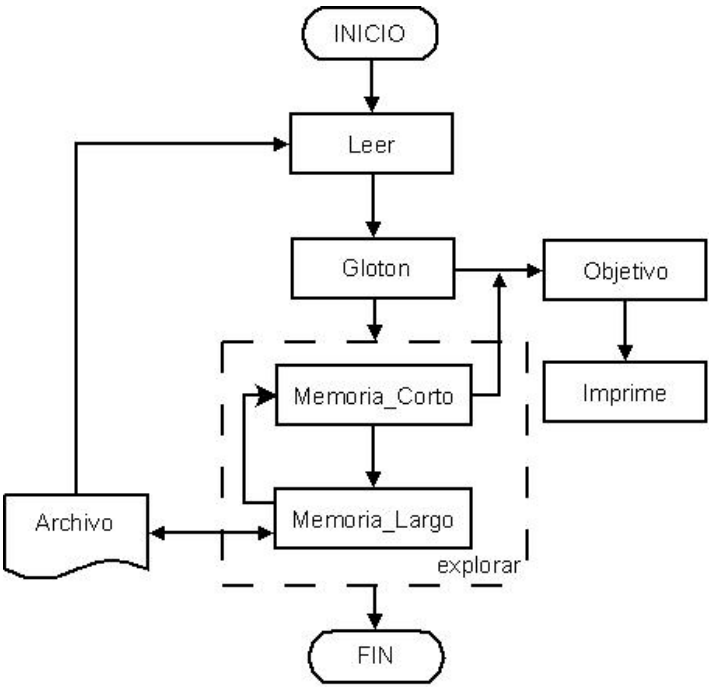

Diagrama 1. Diagrama de flujo de ejecución

Pseudocódigo

Función leer

Propósito: Leer de un archivo de entrada un conjunto de datos que se guardan en una matriz para uso en el programa

Descripción:

Inicializa
Paso 1: Sean i y j enteros
Paso 2: Si el archivo no existe, imprime “No se puede abrir archivo" y termina el programa de lo contrario vaya al paso 3.
Paso 3: LEE de la primera línea del archivo para obtener el numero de vértices y el número de colores. Después lee tanto el numero de filas como de columnas

PARA i=1 HASTA numero de vértices CON INCREMENTO +1
recorre filas
PARA j=1 HASTA numero de vértices CON INCREMENTO +1
recorre columnas
ESCRIBE en el arreglo bidimensional a[i][j]

Función glotón

Propósito: crea el arreglo inicial de agentes propuestos a partir de los datos de la matriz leída y almacenada.

Descripción:

Inicializa
Paso 1: Sean flag, i,j, k enteros
Paso 2: PARA i=1 HASTA número de vértices CON INCREMENTO +1
k=0, flag=0,
Mientras arreglo sea cero y k<=número de colores
k se incrementa en 1 y es flag=0
PARA j=i-1 HASTA j mayor a 0 CON DECREMENTO en 1 recorre
arreglo, SI valor de a[i][j]=1 y arreglo=k
flag=1 e ir al paso 4

Paso 3: SI flag=0
ESCRIBE en la posición i del arreglo el valor de k
Paso 4: SI flag=1
ESCRIBE en la posición i del arreglo un valor aleatorio

Función objetivo

Propósito: Calcula la rigidez y la incompatibilidad del arreglo de agentes propuestos

Descripción

Inicializa
Paso 1: Sean i,j, rig=0, inc=0 enteros
Paso 2: PARA i=1 HASTA número de vértices CON INCREMENTO +1
recorre filas
PARA j=1 HASTA número de vértices CON INCREMENTO +1
recorre columnas
SI el valor del arreglo de coloración en la posición i es
igual al a su posición en j
suma en rig el valor de la posición en el arreglo a[i][j]
suma en inc el valor de la posición en el arreglo a[j][i]

Función imprime

Propósito: imprime el arreglo de agentes propuestos así como la rigidez, incompatibilidad y el total de estos últimos.

Descripción

Inicializa
Paso 1: Sea i entero
Paso 2: PARA i=1 HASTA número de vértices CON INCREMENTO +1 hacer
IMPRIME arreglo de coloración
IMPRIME rigidez, incompatibilidad y la suma de estos

Función Memoria

Propósito: guarda las posiciones que nos den un mejor resultado en rigidez en un arreglo para aplicarles tabú posteriormente

Descripción

Inicializa
Paso 1: Sea x entero, cola[5]
Paso 2: Si tamaño de arreglo de memoria <=5
Agregar valor a la memoria, sino ir a paso 3
Paso 3: PARA x=1 HASTA tamaño arreglo memoria CON INCREMENTO +1 hacer, posición x igualar a x+1 para rotar los datos a la izquierda y sacar primer valor que entro para poder agregar otro valor. Agregar valor a la memoria

Función BuscarTabu

Propósito: busca si la posición a evaluar se encuentra en el arreglo de memoria, si se encuentra ese valor, es tabú

Descripción

Inicializa
Paso 1: Sean re, tab=-1 enteros
Paso 2: PARA re=1 HASTA tamaño arreglo memoria CON INCREMENTO +1 hacer, SI valor de memoria es igual a valor tabú entonces éste valor se guarda en la variable tab regresando ese valor, en otro caso la variable tab regresa -1

Función Mejora_Rig

Propósito: calcula una menor rigidez de un arreglo de agentes propuestos por medio de una posición que recibe siempre y cuando no se encuentre en tabú.

Descripción

Inicializa
Paso 1: Sean val_real,aux enteros
Paso 2: Se utiliza función BuscarTabu para saber si la posición es tabú y el resultado se guarda en la variable aux
Paso 3: mientras aux = posición sino ir a paso 7
Paso 4: Si posición a evaluar < al número de vértices posición se recorre en +1, sino ir a paso 5
Paso 5: Si aux=número de vértices, La posición es igual a 1
Paso 6: volver a usar *BuscarTabu*
Paso 7: guardar la posición del arreglo de coloración en val_real
Paso 8: Escribe en esa nueva posición un valor aleatorio
Paso 9: calcular rigidez e incompatibilidad del arreglo con función objetivo, sino ve a paso 11
Paso 10: SI la suma de rigidez e incompatibilidad es menor al mejor_rig_inc, la suma se escribe en mejor_rig_inc. La posición que arrojó esa mejora entra en función *Memoria*
Paso 11: El valor guardado en val_real se regresa a su posición en el arreglo

Función Memoria_Corto

Propósito: Genera los parámetros requeridos para la función *Mejora_Rig* y busca n veces mejorar el arreglo de coloración en rigidez e incompatibilidad, generando una mejor solución.

Descripción

Inicializa
Paso 1: Sean i, valor, w enteros
Paso 2: PARA i=1 HASTA 100 CON INCREMENTO +1, busca mejor solución
Aleatoriza un color y se guarda en valor
Paso 3: PARA w=1 HASTA num de vértices de arreglo CON INCREMENTO +1
Entran parámetros en *Mejora_Rig*

Función cambios

Propósito: cuantifica las repeticiones de la mejor solución de cada posición de cada corrida y los guarda en una matriz.

Descripción

Inicializa
Paso 1: Sean i,j, arreglo c, enteros
Paso 2: PARA i=1 HASTA número de vértices CON INCREMENTO +1
recorre filas
PARA j=i+1 HASTA número de vértices CON INCREMENTO +1
Recorre columnas
SI posición del arreglo de coloración v[i]= v[j]
Suma lo que hay en la posición c[i][j], con +1 y sobrescribe

Función Tendencia

Propósito: Busca los valores de mayor tamaño en el arreglo c, los cuales se toman en cuanta para formar un nuevo arreglo de agentes propuestos.

Descripción

Inicializa
Paso 1: Sean i,j, maxval=0 enteros
Paso 2: PARA i=1 HASTA numero de vértices CON INCREMENTO +1
Se fija el valor de la fila x y se recorre la columna
SI arreglo c[x][i] > maxval, valor de arreglo c, se guarda en maxval
Paso 3: PARA j=1 HASTA numero de vértices CON INCREMENTO +1
recorre la columna
SI maxval/1.5 < c[x][j] y además el valor del arreglo g[j]=0, se coloca un color en esa posición

Función NuevoGloton

Propósito: Genera los parámetros para la función Tendencia para así asignar un color a una posición a un arreglo que es cero.

Descripción

Inicializa
Paso 1: Sean i, k=0 enteros
Paso 2: PARA i=1 HASTA número de vértices CON INCREMENTO +1
SI valor de arreglo es g=0, se incrementa el valor del color k
Paso 3: Función *Tendencia(c,i,k)* asigna colores al arreglo g

Función Memoria_Largo

Propósito: Recibe el arreglo de agentes propuestos generado de la memoria a corto plazo y guarda en un archivo valores que posteriormente se utilizarán para crear un nuevo arreglo de coloración.

Descripción

Inicializa
Paso 1: Sean i, j enteros;
Paso 2: Se crea un archivo que almacenara una matriz con los datos que proporcionen distintos agentes
Paso 3: Si el archivo no existe, la función *cambios* evalúa el arreglo de agentes propuestos, y los resultados los escribe en el archivo en una matriz de lo contrario vaya a paso 4
Paso 4: Lee el archivo y guarda la información en el arreglo c,
Función *cambios* evalúa el arreglo de agentes propuestos, y los resultados se escriben en el archivo
Paso 5: la función *NuevoGloton* crea en nuevo arreglo de agentes propuestos

Función explorar

Propósito: Es el encargado de buscar n veces una óptima solución utilizando las funciones *Memoria_Corto* y *Memoria_Largo*.

Descripción

Inicializa

Paso 1: Sean i, j, mejor_total=MAX, mejor_rig_inc=(rig+inc)
Paso 2: PARA i=1 HASTA 10 CON INCREMENTO +1, busca solución
Arreglo de coloración es procesado en función *Memoria_Corto*
Al nuevo arreglo de coloración se le calcula incompatibilidad y rigidez con la función *objetivo*

Paso 3: Mientras la incompatibilidad sea diferente de cero. El arreglo de coloración es procesado por *Memoria_Corto*, cuyo resultado se le calcula el objetivo y mejor_rig_inc se inicializa con la finalidad de hacer una exploración más exhaustiva, de lo contrario ir a paso 4

Paso 4: Si (rig+inc)< mejor_total, se guarda (ring+inc) en mejor_total
El arreglo que proporcionó esa mejora es guardado en otro arreglo para que sean comparados en el próximo ciclo

Paso 5: el arreglo mejorado es procesado por *Memoria_Largo*

Paso 6: al final de ciclo, el arreglo de coloración con solución óptima es evaluado por la función *objetivo* y presentada con la función *imprime*

Función main

Propósito: ejecuta el programa y calcula el tiempo de ejecución

Descripción

Inicializa

Paso 1: Sean clock_t t_ini, t_fin, double secs

Paso 2: se obtiene un tiempo del reloj para iniciar

Paso 3: se llama a ejecución (en el orden indicado) las siguientes funciones

Leer
gloton
objetivo
imprime
explorar
objetivo
explorar

Paso 4: se obtiene un tiempo de ejecución final, con la siguiente formula
secs = (double)(t_fin - t_ini) / CLOCKS_PER_SEC
IMPRIME tiempo de ejecución

Implantación

El proyecto fue desarrollado en ANSI C, el cual tiene las siguientes características:

- Funcionalidades añadidas importantes, como funciones matemáticas y de manejo de archivos, proporcionadas por bibliotecas.
- Es un lenguaje muy flexible que permite programar con múltiples estilos según la experiencia o costumbre del programador.
- Impide operaciones sin sentido.
- Acceso a memoria de bajo nivel mediante el uso de apuntadores.

Se Utilizó GNU/Linux (Ubuntu 8.04) como plataforma de desarrollo. Sin embargo, se puede compilar el algoritmo final para que funcione en cualquier otro Sistema Operativo ya que es un estándar del lenguaje C.

Para este trabajo se empleó el compilador gcc (Colección de Compiladores GNU) cuyas características destacan:

- Usa un entorno de desarrollo abierto
- Soporta muchas otras plataformas con el fin de fomentar el uso de un compilador-optimizador de clase global, para atraer muchos equipos de desarrollo, para asegurar que GCC y los sistemas GNU funcionen en diferentes arquitecturas y diferentes entornos, y más aún, para extender y mejorar las características de GCC.

Instalación

En Linux

Para su instalación hay que compilar el proyecto utilizando un IDE[6]. En este trabajo se utilizó el “geany” para Linux. Los pasos para la compilación y ejecución del programa “”Tabú” son:

Se ejecuta geany y abrimos el archivo C que contiene el proyecto, en el menú de opciones tenemos un botón que dice “Compilar”, lo apretamos y en el compilador debe aparecer la leyenda “La compilación termino con éxito” para así tener la seguridad de que el proyecto ha sido compilado. En la siguiente figura muestra la interfaz y la localización de dicho botón para su compilación.

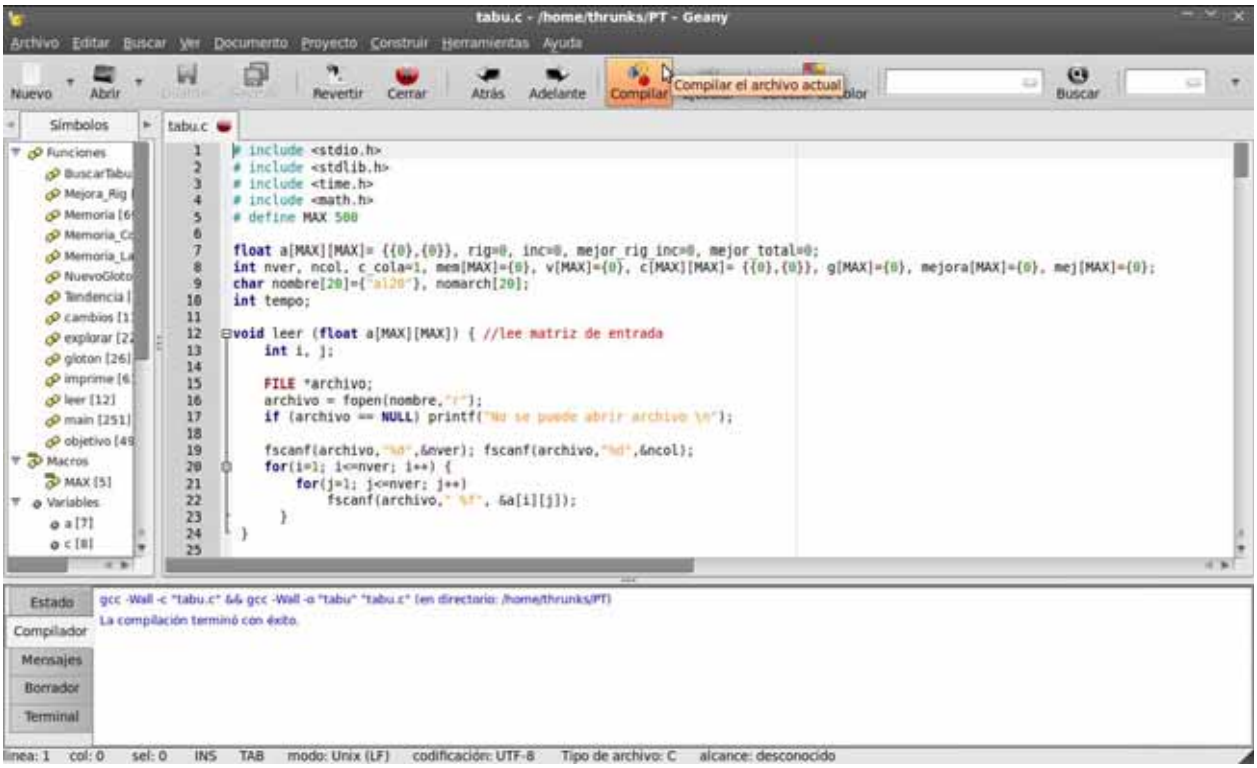

Figura 1 Compilación del proyecto

[6] Entorno de Desarrollo Integrado empaquetado como un programa de aplicación, el cual consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI.

Para su ejecución apretamos el botón “Ejecutar” que se encuentra del lado derecho del botón “Compilar” en el menú de opciones e inmediatamente aparecerá una ventana donde se ejecuta el programa y entrega el resultado.

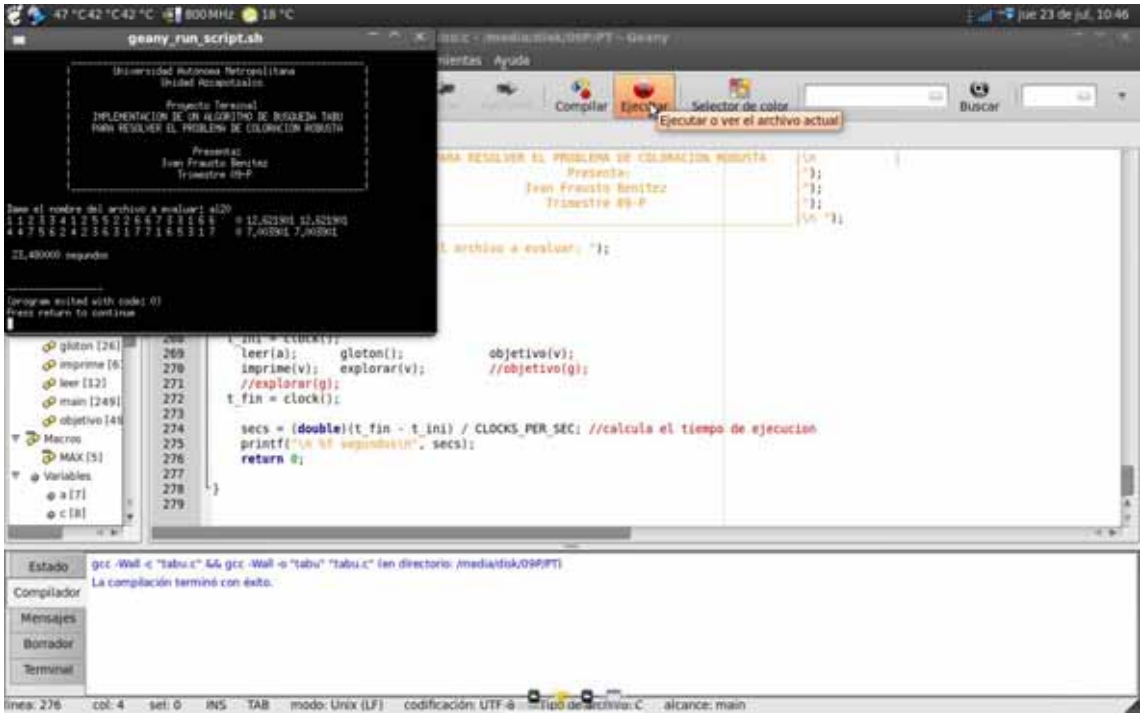

Figura 2. Ejecución del proyecto

Si no se tiene el programa geany se puede compilar y ejecutar desde la consola Linux introduciendo las siguientes líneas de comando:

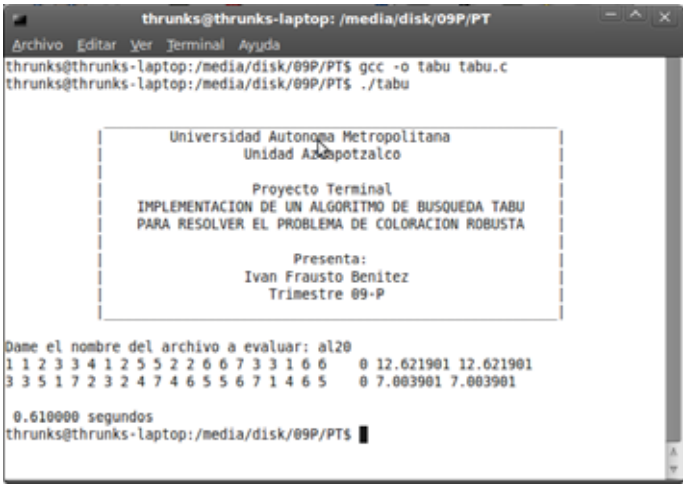

Figura 3 Para su compilación $ gcc –o tabu tabu.c Para su ejecución $ ./tabu

En Windows

Para su instalación y ejecución se requiere de un IDE llamado *Dev-C++* el cual tiene similitud con el programa *geany* de Linux. Se requiere abrir el archivo, compilarlo y esperar a que aparezca una ventana que nos muestre que la compilación no tuvo errores, tal y como se muestra en la siguiente figura

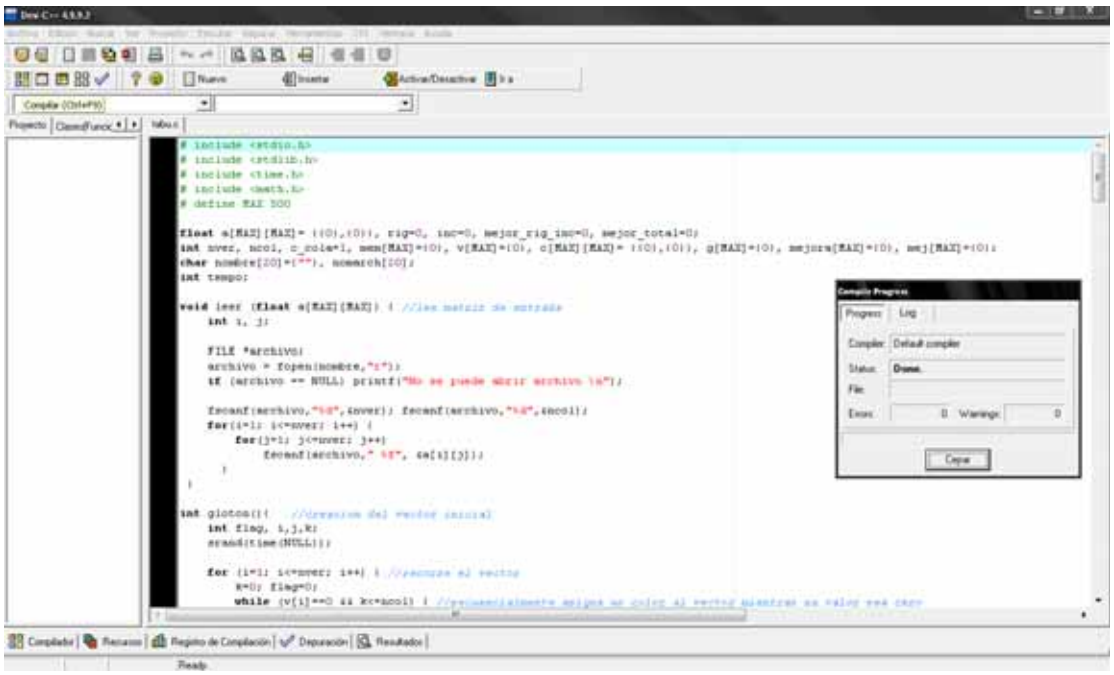

Figura 4 Para su compilación apretar botón que tiene la leyenda compilar

Al tener una compilación exitosa se generará un archivo con extensión *exe* el cual nos permite ejecutar el programa como se muestra a continuación:

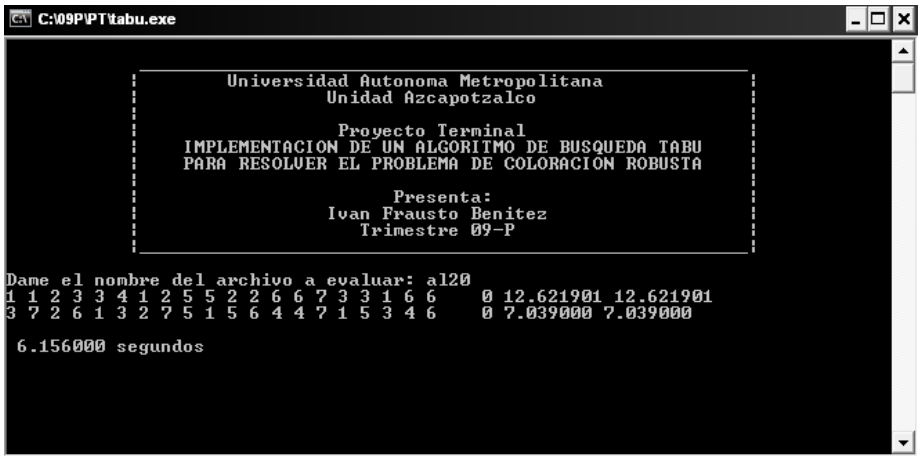

Figura 5 Ejecución del programa en Windows con el archivo "tabu.exe"

Pruebas

Para probar la eficiencia y tiempo promedio de ejecución de este trabajo se utilizaron las siguientes instancias, cuyos resultados fueron comparados con los resultados de un algoritmo Tabu Search[7] y un algoritmo GRASP[8], por lo que se obtuvo lo siguiente.

Instancias			**Rigidez con agentes validos / Tiempo (secs.)**		
$G_{n,0.5}$	n	k	Tabu Search	GRASP	**Tabú**
al(20)	20	7	7.0970 / 2	7.1423 / 0.14	**7.0039 / 4.69**
al(30)	30	10	8.0623 / 5	7.5749 / 0.44	**7.5748 / 7.71**
al(40)	40	14	7.1709 / 15	7.3950 / 1	**7.0868 / 9.58**
al(50)	50	17	9.8259 / 33	8.9531 / 1.9	**8.3539 / 22.5**
al(60)	60	20	9.8331 / 69	9.9687 / 3.3	**9.5987 / 24.61**
al(70)	70	24	11.1307 / 128	11.2388 / 5.3	**10.6420 / 20.80**
al(80)	80	27	11.1946 / 218	11.7512 / 8	12.3387 / 70.02
al(90)	90	30	12.2832 / 350	13.4919 / 11.5	13.3193 / 122.21
al(100)	100	34	12.1932 / 544	12.8675 / 15.8	11.6711 / 41.63
al(110)	110	37	-	12.7681 / 20.7	13.5697 / 103.41
al(120)	120	40	-	15.0014 / 26.8	**14.7455 / 118.54**

Las pruebas realizadas con las instancias conocidas al programa “Tabú”, fueron comparados con dos algoritmos: Un búsqueda tabú (Tabu Search) y un GRASP. Ambos algoritmos dan una solución al problema de coloración robusta y fueron elegidos porque aplican técnicas usadas en la implementación de este trabajo.

[7] Presentado por el Dr. Javier Ramírez, algoritmo mediante el cual a través de una solución al azar, aplica búsqueda local con tabú.

[8] Presentado por el Dr. Pedro Lara, es un algoritmo glotón aleatorizado (donde no escoges la mejor solución, sino que buscas entre las dos o tres mejores y al azar se decide por una). A ese glotón aleatorizado le agregamos una búsqueda local. El algoritmo se corre 100 veces (es decir empieza de 100 soluciones glotonas, en principio diferentes) y toma la mejor de las 100.

Se observa que ofrece mejores soluciones en un 63.63% el programa “Tabú” que los dos algoritmos, debido a que al aplicar la técnica tabú con la técnica de un glotón aplicada en “GRASP” nos proporciona agentes propuestos mas estables y que a partir de estos se realiza la búsqueda local sin tener que explorar áreas de solución factibles. Al Compararlos de manera individual el programa “Tabú”, ofrece soluciones mejores en un 81.81% que el “Tabu Search” y la ejecución se realiza en un tiempo mucho menor, ya que se inicia de una solución glotona y no de una solución al azar. También, da solución a las instancias más grandes de prueba que hasta el momento “El Tabu Search” no le era posible encontrar debido al tiempo tan extenso que requiere para encontrar una posible solución. Con respecto al algoritmo GRASP, el programa “Tabú” ofrece soluciones mejores en un 81.81% aunque no fue posible tener menores tiempos de ejecución en cada instancia de prueba.

Conclusiones

Este trabajo presenta una implementación de un algoritmo de búsqueda tabú como solución al problema de Coloración Robusta. Para su implementación se analizó el estado del arte del problema, de este análisis se propusieron distintas funciones de un programa computacional que proporciona una solución viable a este problema.

Nuestro algoritmo de búsqueda tabú fue implementado en lenguaje C permitiendo obtener ventaja de: 1) su rapidez de ejecución y 2) el paradigma estructurado. Las funciones pueden ser manipulables según las necesidades del problema. Si se utiliza otro lenguaje de programación, como java, aunque se llegase a una solución factible, su tiempo de ejecución promedio sería más alto, ya que es un lenguaje que debe ser interpretado[9].

Para determinar la eficacia, eficiencia y tiempo promedio de ejecución del algoritmo implementado se ejecutó con instancias de prueba conocidas. El resultado de este proyecto terminal es un algoritmo tabú estable que presenta buenos resultados, en un tiempo de ejecución aceptable. Este trabajo alcanza su meta presentando una solución viable al problema de coloración robusta mediante la búsqueda tabú a partir de una solución glotona.

[9] Proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. Se ha prescindido por completo los punteros, y la recolección de basura elimina la necesidad de liberación explícita de memoria.

Trabajo Futuro

Se puede desarrollar un algoritmo de “colonia de hormigas” la cual es una técnica similar a la de búsqueda Tabú. El algoritmo de este proyecto puede servir como base para esta nueva técnica.

Bibliografia

[1] Glover, Fred. “Tabu Search – Part I”, *ORSA Journal on computing* ,v. 1, n. 3, pp. 190-206. Summer 1989

[2] J. Yañez y J. Ramirez, “The robust coloring problem”, *European Journal of Operational Research*, Vol. 148, no. 3, pp. 546-558, 2003.

[3] Ramírez Rodríguez J, *Extensiones del Problema de Coloración de Grafos,* Tesis de Doctorado. Universidad Complutense de Madrid. pp. 26-45, noviembre 2000

[4] Lara Velázquez, Pedro “Un algoritmo evolutivo para resolver el problema de coloración robusta”*, Revista de Matemática*, Vol 12, Nums 1 & 2 pp. 111-120, Junio 2005

[5] M. A. Gutiérrez Andrade, P. Lara Velázquez, S.G. de los Cobos Silva, *A* “new Simulated Annealing algorithm for the Robust Coloring Problem”, *Journal of Industrial Engineering International*, July 2007, Vol 3. Num. 5

[6] GNU Project, GCC, the GNU Compiler Collection,
http://gcc.gnu.org/
Revisada el 31/octubre/2008.

[7] Glover, Fred & Laguna, Manuel *Tabu search*.
http://leeds-faculty.colorado.edu/laguna/articles/ts2.pdf
Revisada el 31/octubre/2008.

[8] Martí, Rafael. *Procedimientos Metaheurísticos en Optimización Combinatoria.*
http://www.uv.es/~rmarti/ pp 24
Revisada el 15/octubre/2008.

[9] Pedro Lara, Lizbeth Gallardo, Miguel Ángel Gutiérrez y Sergio de los Cobos, “Asignación de frecuencias en telefonía celular aplicando el problema de coloración robusta”, *Revista de Matemática*, Vol. 16 Num. 2. correspondiente a julio-diciembre de 2009.