Trabajo práctico de estructura de datos y algoritmos

Coloreo de grafos

Grupo: 5

Alumnos: Jimena Pose (49015), Axel Wassington (50124).

Fecha de entrega: 28 de mayo de 2010.

¿**Para qué sirve el coloreo**?

Para empezar hay un muchos problemas que se pueden resolver con coloreo directamente, por ejemplo:

- El problema de los horarios de los profesores y alumnos, se podría resolver uniendo con aristas las materias que tienen un profesor o un alumno en común; el coloreo daría las materias que se pueden dictar al mismo tiempo.
- El problema de la cadena alimenticia (cuantas jaulas se necesitan para encerrar a una cantidad de especies sin que se coman entre ellas), se podría resolver uniendo con aristas las especies que se comen una la otra; el coloreo diría qué especies pueden estar en la misma jaula.
-

Y así se podrían dar muchos ejemplos similares, donde hay que dividir en grupos que no tengan dos ejemplares que cumplan cierta relación.

Pero lo interesante del coloreo es que es un problema nP-completo. Sat se puede reducir a coloreo, y por ende todos los problemas nP. Así que la lista de problemas que se pueden resolver con coloreo se vuelven rápidamente infinitas. Por ejemplo se podría resolver un recorrido euleriano de un grafo con el coloreo de otro grafo.

Consideraciones tomadas

Durante el desarrollo del trabajo práctico se decidió separar los archivos en carpetas con el fin de facilitar la visualización de los archivos y de mejorar la utilización de los mismos. Se crearon las carpetas “graph”, “exact”, “ts” y “greedy”. Dentro de la carpeta graph es donde se encuentran y se deben poner todos los grafos que se desean colorear. En cuanto al resto de las carpetas, dentro de ellas se puede encontrar el resultado del coloreo según el algoritmo utilizado.

El fin de separar el resultado de los algoritmos en carpetas fue dar la posibilidad de comparar los resultados obtenidos en diferentes algoritmos, ya que si se deja todo en la misma carpeta, por los nombres que se piden en la consigna, al correr un algoritmo sobre un grafo y después correr otro sobre el mismo grafo, el segundo algoritmo pisa los archivos generados por el primero.

Por otro lado, el archivo del árbol generado por el algoritmo exact no se llama “tree.dot”, si no que se llama “[nombre_del_grafo]Tree.dot”, con el mismo fin de poder tener los árboles de diferentes grafos a la vez y poder compararlos entre sí. Finalmente, estos son generados en la carpeta exact, ya que solo provienen de este algoritmo.

Algoritmos

Greedy

```
*Greedy(node)
*
*if(node.visited==true)
*       return
*end if
*
*
*node.color= mínimo color que no tenga un adyacente
*node.visited=true
*
*for(adj in vértices adyacentes)
*       If (adj no se puede colorear con la cantidad de colores utilizados hasta el momento)
*               Greedy(adj)
*       end if
*end for
*
*for(adj in vértices adyacentes)
*       Greedy(adj)
*end for
*end Greedy
```

Exact

El algoritmo de exact tiene un preprocesamiento (incluido al principio) y luego se ejecuta el algoritmo.
Lo que hace el preprocesamiento es colorear un kn que incluya el nodo de comienzo

```
*Preprocesamiento(node)
*
*node.color= mínimo color que no tenga un adyacente
*node.visited=true
*
*for(adj in vértices adyacentes)
*       If (adj no se puede colorear con la cantidad de colores utilizados hasta el momento)
*               adj.color=mínimo color que no tenga un adyacente
*               node.visite=true
*       end if
*end for
*
*node.visited=false
*
*end Preprocesamiento
```

```
*Exact(node)
*
*if(node.visited==true)
*	return
*end if
*
*if(la cantidad de colores usados hasta el momento >= la cantidad de colores mínima encontrada)
*	return
*end if
*
*node.color=mínimo color que no tenga un adyacente
*node.visited=true
*
*for(n in nodeList)
*	Exact(n)
*end for
*
*if(no quedan nodos por visitar)
*	me guardo el coloreo y la cantidad de colores
*end if
*
*node.visited=false
*
*end Exact
```

Tabu search

El tabu search usa una estructura que cada vez que se elije un nodo se incrementa en size/3 una variable que lo representa y al final de cada iteración se decrementan en uno, y siempre y cuando esa variable sea mayor que size/2, ese nodo no puede ser elegido.

```
T=10
time=1000 milisegundos
n=20 iteraciones

*TabuSearch()
*
*while(time)
*	repeat n times
*		node=rand(nodeList)
*		N=Neighbour(node)
*		if(Eval(N)<Eval(actual))
*			actual=N
```

```
*                       node->incremento en size/3
*               else if(rand[0,1)<exp((Eval(N)-Eval(actual)/T))
*                       actual=N
*                       node->incremento en size/3
*               end if
*               if(es mejor coloreo que el mejor que tengo)
*                       me lo guardo
*               end if
*       nodeList->decremento en 1
*       end repeat
*end while
*
*end tabuSearch



*Eval(coloreo)
*
*return la cantidad de colores que uso
*
*en Eval


*Neighbour(node)
*
*color = rand(color)
*
*for(adj in vertices adyacentes)
*       NeighbourAux(adj, color)
*end for
*
*end Neighbour


*NeighbourAux(node, color)
*
*node.vsited=true
*node.color=color
*
*for(adj in vertices adyacentes)
*       if(adj.vidsited==false  y  adj.color==color)
*               adj.color= mínimo color que no tenga ningún nodo adyacente que haya sido visitado
*       end if
*end for
*
*end NeighbourAux
```

Complejidad de los algoritmos

greedy:
Complejidad temporal: O(n)
Complejidad espacial: O(n)

exact:
Complejidad temporal: O(n!)
Complejidad espacial: O(n)

tabu search:
Complejidad temporal: ?
Complejidad espacial: O(n)

Tabla de comparación

[name].graph	greedy	ts	exact
Arbol7	3ms----2colors	1000ms----2colors	4ms----2colors
ciclo4	3ms----2colors	1000ms----2colors	3ms----2colors
ciclo5	3ms----3colors	1002ms----3colors	13ms----3colors
ciclo10	3ms----2colors	1000ms----2colors	4ms----2colors
ciclo11	3ms----3colors	1000ms----3colors	?
ciclo23	4ms----3colors	1000ms----3colors	?
ciclo24	3ms----2colors	1000ms----2colors	6ms----2colors
H10,17	17ms----11colors	10002ms----9colors	?
H11,17	22ms----9colors	1000ms----9colors	?
H4,6	3ms----3colors	1000ms----3colors	4ms----3colors
K10	3ms----10colors	1001ms----10colors	4ms----10colors
noGreedy	3ms----5colors	1000ms----4colors	84ms----4colors
randomGraph10	3ms----7colors	1000ms----7colors	4ms----7colors
randomgraph11	3ms----5colors	1000ms----5colors	833ms----5colors
randomgraph20	12ms----7colors	1000ms----7colors	?
randomgraph15	5ms----6colors	1001ms----5colors	?
randomgraph13	4ms----6colors	1002ms----5colors	180695----5colors

Conclusión

A partir de esta tabla de comparación queda claro que la mejor opción es usar el tabu search. Para empezar, en todos los casos estudiados el exact nunca dio un mejor resultado que el tabu search, a lo que se le suma que para grafos con más de 10 vértices, a excepción de algunos casos especiales, tarda demasiado en resolver el coloreo.

Otro aspecto importante es que en muchos casos se puede observar una diferencia de un color entre tabu search y greedy, incluso en el H10,17 la diferencia es de dos colores. Por último, en cuanto a la complejidad temporal y espacial, no hay ninguna desventaja en usar el tabu search, ya que la complejidad espacial es la misma para los 3 algoritmos y el tiempo que tarda el tabu search puede ser cambiado en el código (es un tiempo fijo), siempre en cuando se tenga en cuenta el costo que tiene hacerlo. En el caso de la tabla se decidió correr el algoritmo con 1000 ms.