Informe de Taller 2:
Vertex Coloring Problem

Nombre: Ignacio Solar Arce
Profesor: Pablo Román Asenjo

Sección: B-2
Fecha: 11/12/2024

1. Explicación breve del Programa

El programa desarrollado implementa una solución al problema de coloreo de grafos utilizando **Branch and Bound** complementada con una **Heuristica de Peligro**. El objetivo es asignar la menor cantidad de colores a los vértices de un grafo con la condicion de que no existan vértices adyacentes con el mismo color asignado.

2. Heuristicas y Optimización

2.1. Branch and Bound

El algoritmo principal utilizado es **Branch and Bound**, una técnica de búsqueda que explora las posibles soluciones dividiendo el problema en subproblemas más pequeños (branching) y descartando partes del universo de búsqueda que no pueden contener una solución mejor (bounding). Se utilizan cotas superior e inferior para limitar el rango de colores posibles, reduciendo así el universo de búsqueda y permitiendo encontrar una solución en un tiempo razonable.[1]

2.2. Heuristica de Peligro

Para la selección del siguiente nodo y del siguiente color, se utilizan las formulas de la heuristica de peligro, que llevan a un valor numero, "que tan peligroso es que este nodo no sea coloreado ahora ƨ ademas "cual es el color que tengo disponible que tiene menos peligro para colorear". La efectividad del peligro radica en la combinación de distintos factores ponderados por constantes que fueron determinadas empíricamente. [1]

El peligro de un vértice se ve representado por:

$$danger(i) = \frac{C}{(max\_color - y)^k} + k_u \cdot uncolored(i) + k_a \cdot \frac{share(i)}{avail(i)}$$

Donde:

$$C = 1{,}0$$
$$k = 1{,}0$$
$$k_u = 0{,}025$$
$$k_a = 0{,}33$$

Ademas, el peligro de un color se ve representado por:

$$danger(j) = \frac{k_1}{(max\_color - diff\_neighbors(c))^{k_2}} + k_3 \cdot uncolored(n_c) - k_4 \cdot num(c)$$

Donde:

$$k_1 = 1{,}0$$
$$k_2 = 1{,}0$$
$$k_3 = 0{,}5$$
$$k_4 = 0{,}025$$

Donde las variables representan:

- max_color: Número máximo de colores permitidos
- y: Número de colores diferentes asignados a los vecinos del nodo
- $uncolored(i)$: Número de vecinos sin colorear del nodo i
- $share(i)$: Número de colores disponibles para el nodo i que también están disponibles para sus vecinos sin colorear
- $avail(i)$: Número total de colores disponibles para el nodo i
- $diff\_neighbors(c)$: Número máximo de vecinos con colores diferentes sobre todos los nodos no coloreados que tienen disponible el color c
- n_c: Nodo que alcanza ese máximo
- $num(c)$: Número de veces que el color c ha sido usado

2.3. Calculo de cotas

El programa calcula **cotas superior e inferior** para el número de colores necesarios, esto permite reducir considerablemente el universo de opciones distintas que se tienen que revisar:

- **Cota Superior**: Se calcula utilizando un enfoque analogo al *Largest Degree First* (LDF), asignando colores a los vértices en orden por su grado y decidiendo si se agregan mas colores o no cada vez que se consulta por un vértice.
- **Cota Inferior**: Se estima mediante una *clique máxima* en el grafo, encontrado de manera golosa, ya que una consecuencia de un clique es que la cantidad de colores no puede ser menor que el tamaño de esta misma clique.

3. Aspectos de implementación y eficiencia

3.1. Policy-Based Data Structures

Para ciertas estructuras de datos, se decidió optar por utilizar contenedores de GNU [2] que ofrecen estructuras optimizadas, en especial los 'unordered_map', que debido a que cada contenedor oficial de STL debe seguir las reglas de implementación especificas, genera un hit en performace, ya que los contenedores de GNU no se rigen por las mismas reglas, se obtiene como resultado una estructura de datos análoga a la existente pero que tiene mejor performance. Se utiliza específicamente en la implementación del grafo, donde solo se manejan valores enteros, permitiendo tener un mayor boost de performance.[2]

4. Ejecución del código

Para la ejecución del código se debe utilizar el makefile asociado para compilar el programa, existe las opciones: clean y all. Una ejecución completa se vería de esta forma:

- ejecutar `make clean all`
- ejecutar ./graph
- Se llegara al menú de selección, ingresar archivo, solucionar, ejecutar los tests o salir.
- En el caso de ingresar un archivo, los de ejemplo se encuentras dentro de la carpeta examples, para usar se debe agregar el prefijo examples/file.txt
- Luego se decide solucionar, si es que se puede solucionar el programa retorna la solución, primero unos prints de debug, luego los nodos y el color asignado a cada uno y por ultimo el tiempo total y la cantidad de colores total.

Referencias

[1] F. Glover, M. Parker, and J. Ryan, "Coloring by tabu branch and bound," *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 00, no. 0000, mar 1995, primary 90C35, 68R10; Secondary 05C85.

[2] GNU Project, "GNU C++ Policy-Based Data Structures," https://gcc.gnu.org/onlinedocs/libstdc++/ext/pb_ds/, 2024, accessed: 2024-03-11.