Assignement 3

Iván Piña Arévalo
ivan.pinaarevalo@alum.uca.es

May 10, 2019

Abstract

En esta práctica, realizaremos la implementación de una reducción polinomica. Concretamente, reduciremos el problema del coloreado de un grafo al problema 3-SAT. Una vez implementado nuestro programa, realizaremos análisis sobre su complejidad termporal, así como las relaciones entre el número de colores necesarios y el máximo clique del grafo. Se ha empleado el grafo de Petersen.

1 Introduction

En primer, lugar, comenzaremos hablando del coloreado de un grafo. Este problema consiste en colorear un grafo de tal manera que dos vértices adyacentes no pueden tener el mismo color.

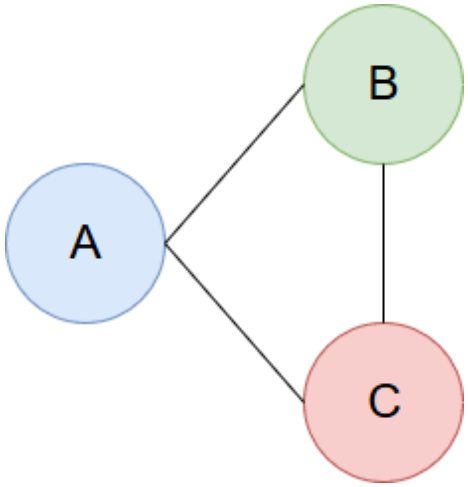

Figure 1: Grafo de ejemplo con una solución válida

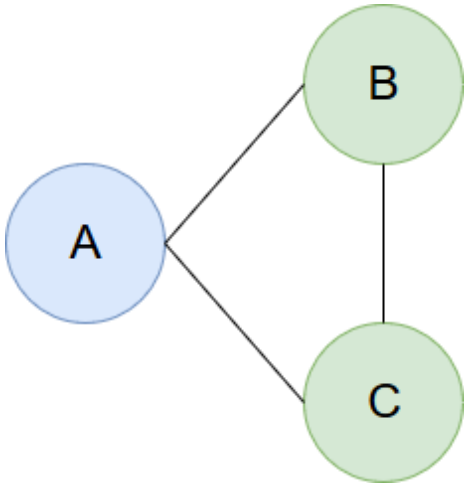

Figure 2: Grafo de ejemplo con una solución inválida

Concretamente, hemos empleado el grafo de Petersen. Hemos variado ligeramente la numeración de los nodos. Para mayor comodidad, el primer nodo lo

hemos numerado como '1' en lugar de '0' y así sucesivamente (es decir, hemos incrementado en 1 el valor de cada nodo.)

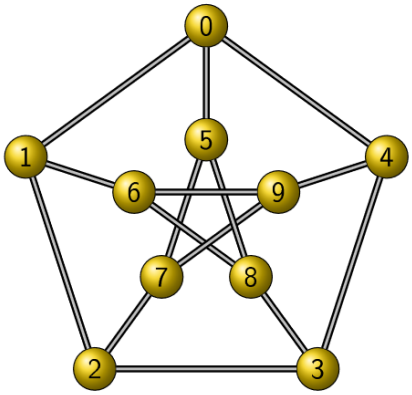

Figure 3: Grafo de Petersen

Para encontrar la solución de este problema se ha empleado el problema 3-SAT. El problema SAT fue uno de los primers en descubrirse NP-completo. Dicho problema consiste en dada una ecuación booleana, encontrar los valores para los cuales es verdadera. Además, la ecuación debe tener las siguientes características:

- La ecuación global se compone de ecuaciones enlazadas con la puerta logica and
- Cada subecuación esta compuesta por variables $(x_1, x_2, ...)$ unidas por la puerta lógica OR.

A continuación podemos ver una ecuación para el problema SAT:

$$(x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee x_4)$$

2 Methodology

La implementación de la práctica se ha realizado en C++, hemos dividido el programa en varias partes.

2.0.1 Lectura del grafo

Para la lectura del grafo nos hemos apoyado en un fichero. Dicho fichero debe tener la siguiente estructura:

- La definición de cada nodo se realiza en una fila.

- Los nodos a los que esta conectado el nodo que estamos definiendo estan separados por comas.

De esta manera realizamos la lectura del grafo, representado mediante listas de adyacencia. A continuación podemos ver el fichero con el grafo de ejemplo de la sección anterior.

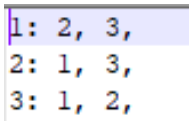

Figure 4: Fichero de entrada para el grafo de ejemplo

De igual manera, el grafo de Petersen representado mediante listas de adyacencia.

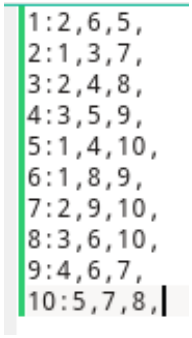

Figure 5: Fichero de entrada para el grafo de Petersen

2.0.2 Transformación del 3-Col to SAT

Lo primero al realizar esta parte ha sido analizar profundamente la estructura de las ecuaciones booleanas que representan al grafo que deseamos colorear. Estas ecuaciones se pueden dividir en dos grupos.

- Ecuaciones correspondientes al color del propio nodo.
- Ecuaciones correspondientes al color de los nodos adyacentes al nodo actual.

Si nos fijamos, en los grafos que hemos empleado las aristas son bidireccionales. Bastaría con definirlas una sola vez, sin embargo, para mantener la legibilidad del código he decicido permitir que se escriban dos veces (una en cada sentido). Este aumento en número de ecuaciones no afecta a PicoSAT. A continuación mostramos la salida para el grafo de Petersen

2.0.3 Escritura de las ecuaciones booleanas

Finalmente es necesario escribir en un fichero las ecuaciones correspondientes al grafo. Estas ecuaciones deben ir en formato DIMACS. El fin de este formato es el correcto procesamiento por parte del programa PicoSAT.

Simplemente en lugar de ir mostrando las ecuaciones por la salida estándar las hemos dirigido hacia una cadena. Igualmente hemos definido una variable que se encarga de contar cuantas ecuaciones tiene y otra más para el número de variables. Volcando las cadenas y los contadores en un fichero conseguimos tener la información lista para PicoSAT.

El programa ha sido paramétrizado de tal forma que tanto el grafo como el número de colores pueda modificarse.

2.0.4 PicoSAT

Finalmente, para ejecutar el programa hemos empleado la orden:
./picosat salida

Si deseamos ver todas las soluciones
./picosat –all salida

3 Results and Discussion

Para averiguar el tipo de reducción obtenida hemos realizado un estudio de la complejidad temporal del programa 3SAT (es decir, suponiendo que el numero de colores es 3).

```
//Each node must take only 1 color
for (size_t i = 1; i <= num_var; i = i + colores)
{
    for (size_t j = 0; j < colores; ++j)
        ss << i + j << " ";
    ss << "0" << endl;
    ++ecuaciones;
    for (size_t j = i; j <= colores + incremento * colores; ++j)
    {
        for (size_t z = j + 1; z <= colores + incremento * colores; ++z)
        {
            ss << "-" << j << " " << "-" << z << " " << 0 << endl;
            ++ecuaciones;
        }
    }
    ++incremento;
}

int left, right;
//Represent connections between nodes
for (size_t i = 0; i < nodos.size(); ++i)
{
    for (size_t j = 0; j < conexiones[i].size(); ++j)
    {
        //Check nodos[i] conexiones[i][j]
        left = nodos[i];
        right = conexiones[i][j];
        for (size_t z = 1; z <= colores; ++z) {
            ss << "-" << z + (left - 1)*colores << " -" << z + (right - 1)*colores
                << " 0" << endl;
            ++ecuaciones;
        }
    }
}
```

Figure 6: Análisis temporal del algoritmo

Como se puede observar, existen dos anidamientos de bucles for. Por tanto, hemos decidido dividir en análisis en dos bloques, cada uno correspondiente a un bloque de bucles for.

Analizando el comportamiento de los bucles, vemos que el número de iteraciones depende de dos factores:

- Número de nodos: n.
- Número de colores: en este caso es 3.

En primer anidamiento distinguimos 4 bucles for:

- for(size_t i = 1; i $\leq$ num_var; i = i + colores): Este bucle se ejecuta n veces
- for(size_t j = 0; j $<$ colores; ++colores): Este bucle se ejecuta colores veces
- for(size_t j = 0; j $\leq$ colores + incremento*colores; ++j): Este bucle se ejecuta colores veces

- for(size_t z = j+1; z $\leq$ colores + incremento*colores; ++z): Este bucle se ejecuta j-1 veces

Por tanto, la complejidad de la primera parte es

$$t(n) = n(3 + 3 * 1) = n$$

Pasamos a analizar la segunda parte del algoritmo. Distinguimos los siguientes bucles

- for(size_t i = 0; i ¡ nodos.size(); ++i): Se ejecuta n veces
- for(size_t j = 0; j ¡ conexiones[i].size(); ++j): Se ejecuta tantas veces como aristas tenga el grafo. En el caso peor (n-1) veces.
- for(size_t z = 1; z ¡= colores; ++z): Se ejecuta 3 veces.

Por tanto, la complejidad de la segunda parte es:

$$t(n) = n((n-1) * 3) = n^2$$

Fusionando ambas partes, por la regla del máximo tenemos que el algoritmo tiene un orden $O(n) = n^2$, es decir, posee un orden polinómico. Nuestro programa posee una complejidad polinómica. Consecuentemente hemos obtenido una reducción polinómica. Empleando tres colores, obtenemos las ecuaciones:
Una solución para el grafo de Petersen es esta:

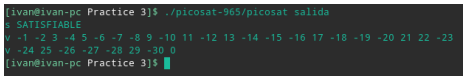

Figure 7: Fichero de entrada para el grafo de Petersen

Así mismo, todas las posibles soluciones son:

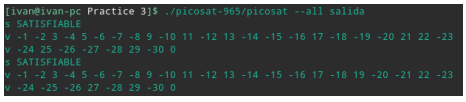

Figure 8: Fichero de entrada para el grafo de Petersen

En cambio, no existe ninguna solución si decrementamos el numero de colores a 2. Finalmente, analizando la estructura del grafo tenemos que el Clique Máximo del grafo de Petersen es de tamaño 2.

El Clique máximo del grafo es 2. Así mismo no podemos colorear el grafo con menos de tres colores. Esto, intuitivamente nos sugiere que

$$clique < numero_colores$$

Siendo clique el numero de nodos que forman el máximo clique y numero_colores el número de colores mínimo necesario para colorear el grafo. Ahora a fin de contrastar esta suposición, decidimos probar con el grafo de ejemplo que hemos mostrado al principio del report.

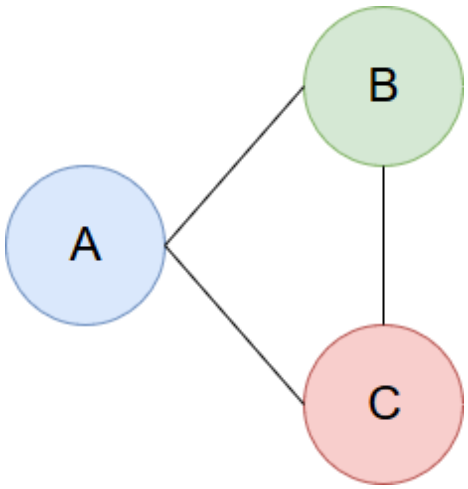

Figure 9: Grafo de ejemplo

Aquí, sin embargo, el clique máximo del grafo es 3 y se puede colorear correctamente con tres colores. Por tanto tenemos que:

$$clique \leq numero_colores$$

Consecuentemente, realizamos un intento de colorear el grafo de Petersen con dos colores

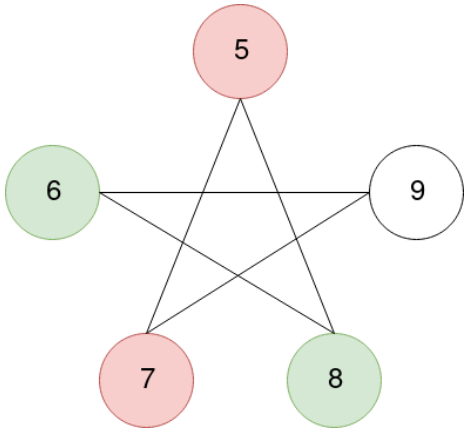

Figure 10: Interior del grafo de Petersen

No ha hecho falta colorear todo el grafo, trabajando únicamente con el interior podemos observar que no es coloreable con sólo dos colores, el nodo numero 9 no puede tomar ni rojo ni verde.

4 Conclussion

Como hemos podido comprobar, se puede realizar una reducción polinomica del problema del coloreado de un grafo a SAT. Este hecho afirma que los problemas NP-Complete contienen a todos los problemas perteneciente a la clase NP.

Por otra parte, respecto a la relación entre el Clique y el número máximo de colores, se llega a la conclusión que no solo el clique influye en el número mínimo de colores necesario. También lo hace la estructura del grafo. Si quereremos ceñirnos estrictamente al Clique para estimar el número de colores, tenemos que

$$clique < numero_colores$$

Aunque en algunos casos, cuando la estructura del grafo lo permita, es posible que se cumpla:

$$clique \leq numero_colores$$

Consecuentemente, si existiera en el grafo un clique de tamaño 4, no sería posible colorear el grafo con únicamente 3 colores. Este hecho es fácilmente razonable por otra parte ya que si existe un clique de tamaño 4, existen vértices (al menos 4), que tendrán tres vértices adyacentes.

5 Bibliografia

- Kleinberg, Jon; Tardos, Éva Tardos (2006). Algorithm Design. Pearson Education. pp. 452–453. ISBN 978-0-321-37291-8.
- Goldreich, Oded (2008), Computational Complexity: A Conceptual Perspective, Cambridge University Press, pp. 59–60, ISBN 9781139472746
- Garey, Michael R.; Johnson, D. S. (1979), Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman