Assignement 3

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

May 10, 2019

Abstract

In this practice, we will make polynomial reduction implementation. Specifically oder in particular, we will reduce the coloured graph to the 3-SAT problem. Once implemented our problem, we will make analysis about temporal complexity, as well as relations between the number of colors required and the Maximum Click of the graph. The Petersen graph has been used.

1 Introduction

First, we start talking about coloring of graph. This problem consists in coloring a graph in such a way that two adjacent vertices cannot have the same color.

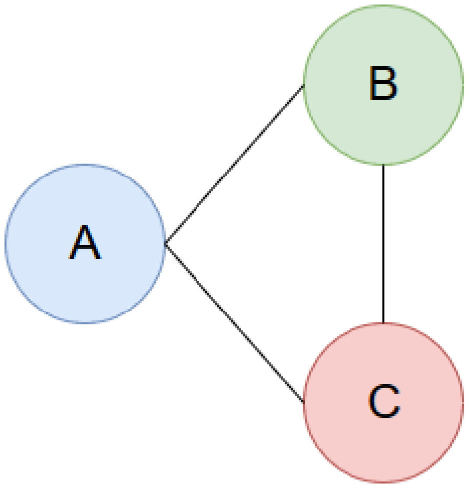


Figure 1: Example with a valid solution.

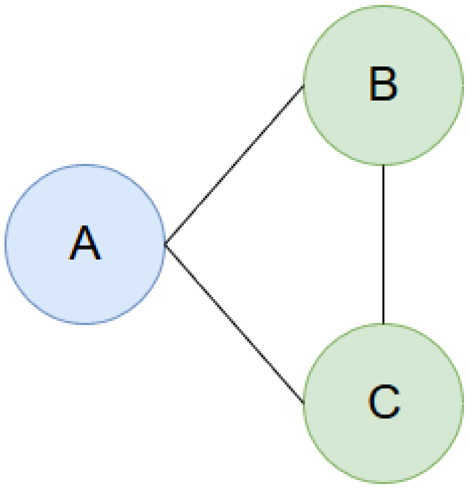


Figure 2: Example graph with a invalid solution

In particular, we had used the Peterson graph. We have slightly varied the numberin of nodes. For convenience, the first node having been numbered ”1”, instead of ”0” and so on (in other words, we have increased to 1 the value of each node).

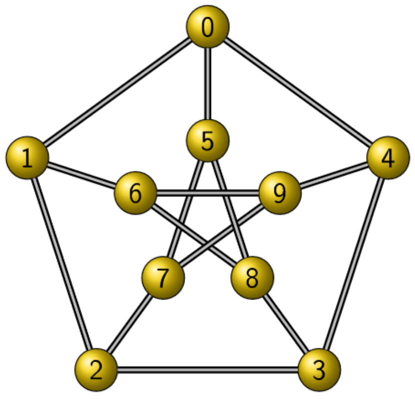


Figure 3: Petersen Graph

To find the solution of this problem, has been used the 3-SAT problem. The SAT problem was one of the first to discover NP-complet. This problem consists of given a Boolean equiation, finding the values for which it is true. In addition, the equiation must have the following characteristics:

- The global equation is composed of equiations linked to the logical gate AND.
- Each subequation is composed of variables $(x_1, x_2, ...)$ joined by the logical door OR.

Below we can see an equation for the SAT problem:

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

2 Methodology

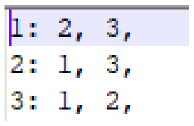
All implementations have been made in C++, we have divided the problem into several parts.

2.0.1 Reading of the graph.

For the reading of the graph we have relied on a file. This file must have the following structure:

- The definition of each node is done in a row.
- The nodes to which the node we ar defining is connected are separated by commas.

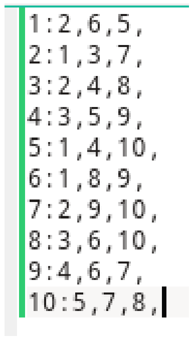
In this way we perform the reading of the graph, represented by adjacency lists. Next we can see the file wiht the example graph of the previous section.



```
1: 2, 3,
2: 1, 3,
3: 1, 2,
```

Figure 4: Input file for the example graph

Similarly, the Petersen graph represented by adjacency lists.



```
1:2,6,5,
2:1,3,7,
3:2,4,8,
4:3,5,9,
5:1,4,10,
6:1,8,9,
7:2,9,10,
8:3,6,10,
9:4,6,7,
10:5,7,8,
```

Figure 5: Input file for the Petersen graph

2.0.2 Transformation 3-Col to SAT

The first thing to do in this part was to deeply analyze the structure of the Boolean equiations that represent the graph we want to color. These equations can be divided into two groups.

- Equations corresponding to the color of the node itself.
- Equations corresponding to the color of the nodes adjacent to the current node.

If we look at the graphs we have used the edges are bidirectional. It would suffice to define them only once, however, to mantain the readability of the code I have decided to allow them to be written twice (one in each direction). This increase in number of equations does not affect PicoSAT. Next we show the output for the Petersen graph.

2.0.3 Writing boolen equations

Finally, it is necessary to write in a file the equations corresponding to the graph. These equations should be in DIMACS format. The purpose of this format is the correct processing by the PicoSAT program.

Simply instead of showing the equations by the standard output we have directed them towards a string. We have also defined a variable that is responsible for counting how many equations it has and another variable for the number of variables. Turning the strings and the counters in a file we get to have the information ready for PicoSAT.

The program has been parametrized in such a way that both the graph and the number of colors can be modified.

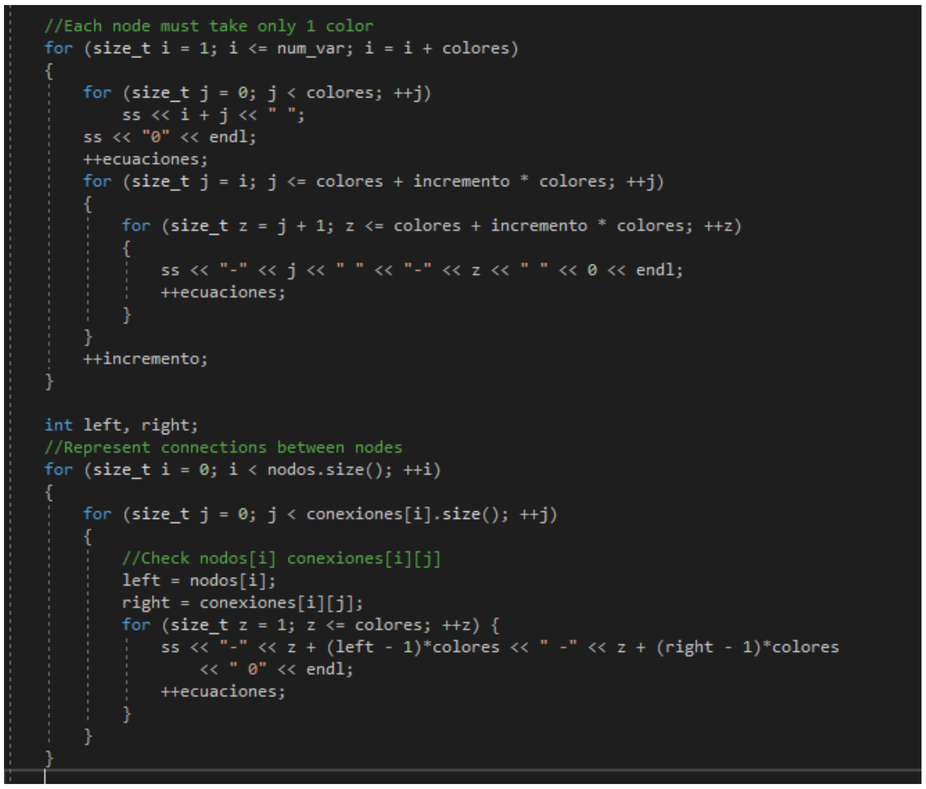
2.0.4 PicoSAT

Finally, to execute the program we have used the orde:
./*picosat* salida

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

3 Results and Discussion

To find out the type of reduction obtained, we have made a study of the temporal complexity of the 3SAT program (that is, assuming the number of colors is 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: Temporal analysis of the algorithm

As you can see, there are two nests of for loops. Therefore, we have decided to divide in analysis into two blocks, each corresponding to a block of for loops.

Analyzing the behavior of the loops, we see tat the number of iterationes depends on two factors:

- Number of nodes: n.
- Number of colors: in this case it is 3.

At first nesting we distinguish 4 loops for:

- for(size_t i = 1; i $\leq$ num_var; i = i + colores): this loop is executed n times.
- for(size_t j = 0; j < colores; ++colores): this loop is executed colors times
- for(size_t j = 0; j $\leq$ colores + incremento*colores; ++j): this loop is executed colors times
- for(size_t z = j+1; z $\leq$ colores + incremento*colores; ++z): this loop is executed j-1 times

Therefore, the complexity of the first part is:

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

We now analyze the second part of the algorithm. We distinguish the following loops:

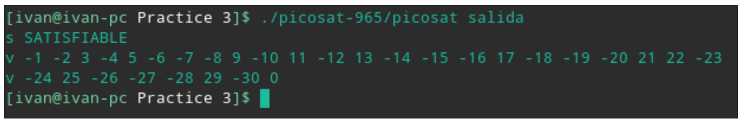
- for(size_t i = 0; i ¡ nodos.size(); ++i): it is executed n times.
- for(size_t j = 0; j ¡ conexiones[i].size(); ++j): it is executed as many times as edges have the graph. In the worst case (n-1) times.
- for(size_t z = 1; z ¡= colores; ++z): it is executed 3 times.

Therefore, the compexity of the second part is:

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

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.

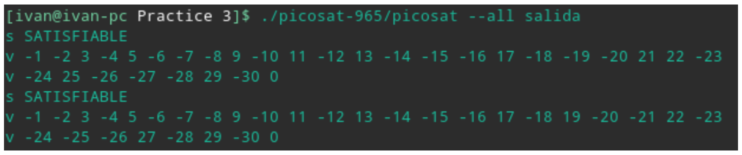
A solution for the Peterson graph with three colors is this:



```
[ivan@ivan-pc Practice 3]$ ./picosat-965/picosat salida
s SATISFIABLE
v -1 -2 3 -4 5 -6 -7 -8 9 -10 11 -12 13 -14 -15 -16 17 -18 -19 -20 21 22 -23
v -24 25 -26 -27 -28 29 -30 0
[ivan@ivan-pc Practice 3]$
```

Figure 7: Solution for the Petersen graph

Likewise, two solutions are:



```
[ivan@ivan-pc Practice 3]$ ./picosat-965/picosat --all salida
s SATISFIABLE
v -1 -2 3 -4 5 -6 -7 -8 9 -10 11 -12 13 -14 -15 -16 17 -18 -19 -20 21 22 -23
v -24 25 -26 -27 -28 29 -30 0
s SATISFIABLE
v -1 -2 3 -4 5 -6 -7 -8 9 -10 11 -12 13 -14 -15 -16 17 -18 19 -20 -21 22 -23
v -24 -25 -26 27 -28 29 -30 0
```

Figure 8: Two solutions for the Petersen graph

On the other hand, there is no solution if we decrease the number of colors to 2. Finally, analyzing the structure of the graph we have that the Maximum Clique of the graph of Petersen is of size 2.

The Maximum Clique of the graph is 2. Furthermore we can not color the graph with less than three colors. This, intuitively suggests that:

$$clique < numero\_colores$$

By clique the number of nodes that form the maximum clique and number_colors the minimum number needed to color the graph. Now in order to contrast this assumption, we decided to try the sample graph we have shown at the beginning of the report

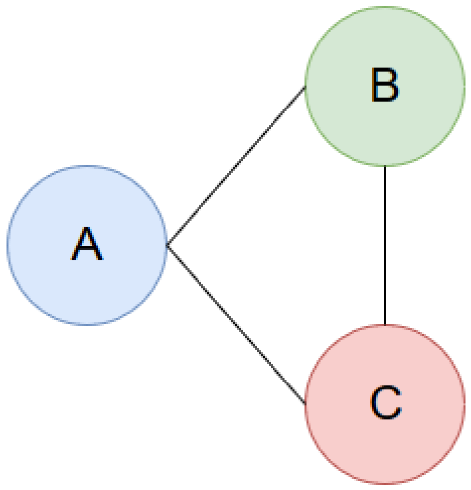


Figure 9: Example graph

Here, however, the maximum clique of the graph is 3 and it can be colored correctly with three colors. Therefore we have to:

$$clique \leq numero\_colores$$

Consequently, we made an attemplt to color the Petersen graph with two colors

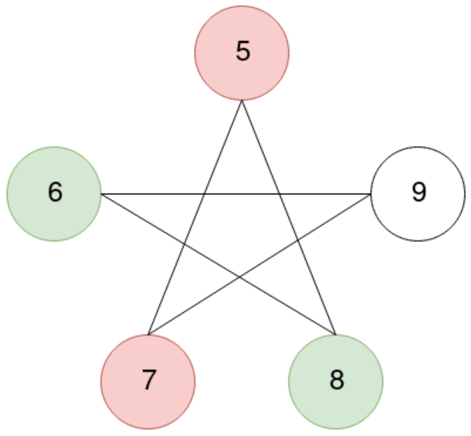


Figure 10: Interior del grafo de Petersen

It was not necessary to color the whole graph, working only with the inside we can see that it is not colorable with only two colors, the node number 9 can not take neither red nor green.

4 Conclussion

As we have been able to verify, a polinomic reduction of the problem of the coloring of a graph to SAT can be realized. This fact states that the NP-Complete problems contain all the problems belonging to the NP class.

On the other hand, regarding the relation between the Clique and the maximum number of colors, it is concluded that not only the Clique influences the maximum number of colors needed. So does the structure of the graph. If we want to stick strictly to the Clique to estimate the number of colors, we have to:

$$clique < numero\_colores$$

Although in some cases, when the structure of the graph allows it, it is possible that be fulfilled oder be accomplished:

$$clique \leq numero\_colores$$

Consequently, if there was a Clique of size 4 in the graph, it would not be possible to color the graph with only 3 colors. This fact is easily reasonable, on the other hand since if there is a Clique of size 4, there are vertexes (al least 4), which will have three adjacent vertexes.

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