Assignment 3

Rubén Pérez Mercado

December 4, 2022

1 Introduction

In this assignment, we will solve the Clique problem by reducing it to the CNF Satisfiability Problem. We will solve both the optimization and the search problem using *Picosat*.

1.1 Definitions

1.1.1 Clique

Let $G = (V, E)$ be an undirected graph. $S \subseteq V$ is a clique of G if all the vertexes in S are adjacent to each other. S induces a complete subgraph of G. A k-clique is a clique of size k

1.1.2 The Clique Problem

- Decision Version: Given $1 \leq k \leq n$, does G contains a k-clique?
- Search Problem: Given $1 \leq k \leq n$, find a k-clique of G, if any.
- Optimization Problem: Find a maximum-size clique of G.

1.1.3 CNF Satisfiability Problem

The CNF Satisfiability Problem (CNF-SAT) is a version of the Satisfiability Problem, where the boolean formula is specified in the Conjunctive Normal Form (Conjunction of Disjunctions of Literals). A literal is a variable or its negation. The disjunctions are called clauses.

2 Methodology

For this assignment, we have created five different files. In Section 2.1 we will see the reducer. Section 2.2 shows the auxiliary class we created to represent our graph. Section 2.3 will detail how we have solved the optimization problem and the search problem. In Section 2.4, the code that stores the solutions in files is explained. Finally, in Section 2.5 we will see the main function of this program.

2.1 Reducer

We have implemented our reducer in “reducer.hpp”.

This method transforms our graph (previously generated in another file) to a CNF instance in DIMACS format. This format has a clause per line, if the values are negative, we refer to the negated variable. Each line ends in “0”. We will create all the clauses, and we will encode our literals in the following way:

$$x_{ir} = r * n + i$$

Where n is the number of nodes in our graph.

2.2 Graph Class

We have used our graph class from Assignment 2. It is implemented in “graph.hpp”.

This class has helped us to create graph instances out of files in DIMACS format. The reduction has been so much easier, as we have used the constructor we created in the last assignment.

However, we have changed some things in this class. First, we have removed the methods and attributes concerning the order of the vertexes. In addition, we have created two new methods, that we will use to modify our graph in the search problem:

- removeVertex: Removes the vertex of a graph. In fact, the vertex is not being removed, but all the connections of this vertex are. Returns the nodes that were connected with that vertex.
- addVertex: The complementary operation of removeVertex. It needs a vector with all the connections to restore.

2.3 Solver

The code of both solver functions is in “solver.cpp”.

We have implemented in this file the algorithm to solve the search problem and the optimization problem by using *Picosat* as an oracle that solves the decision problem of CNF Satisfiability.

2.3.1 Search Problem

For the search problem, we reduced the graph to a CNF-SAT instance and make a query to our oracle. If there are no solutions for the reduced instance of the original graph, it means that there are no k-cliques.

If the oracle says that there is a solution, we will start removing vertexes from our graph, reducing the resulting graph to a CNF-SAT instance and making another query. We will have now two possibilities:

- There are no solutions for this subgraph. It means that the vertex we removed was part of our k-clique. We add again this vertex, and continue with the rest of the vertexes.
- There is at least a solution for this subgraph. In this case, it means that the vertex we removed was not part o a k-clique, or it was part of a bigger clique. In both cases, we can get rid of this

vertex, as we know there is at least one solution left in the subgraph. We continue with the rest of the vertexes.

We will repeat this process until we find all the vertexes that are in our k-clique. At this point, we remove all the remaining connections, and return the solution.

In the special case of $k = 1$, we know there is going to be a solution. In this case, we will loop until we reach the second last vertex, leaving the graph without any connections.

With this algorithm, the maximum number of oracle calls is $n + 1$, and the minimum number (if there is a solution) is $k + 1$. In any case, the total number of oracle calls is polynomially-bounded.

2.3.2 Optimization Problem

In this case, we have to do a binary search in the space of possible values. As we are not assuming that the graph has edges and the number of vertexes has to be at least 1, we are going to do our search in the space of values $[1, n]$. Just by making "log(n)" oracle calls (and $log(n)$ reductions from graph to CNF-SAT instances, changing the value of k), we are able to find the optimal value. When we get that optimal value, we call the search problem solver (we have called the search problem solver because the process we have to follow here is the same as in the search problem).

2.4 Writer

The writer function (implemented in “writer.hpp”) will store the subgraphs that the solver functions return in a file (with DIMACS format). It receives a graph instance.

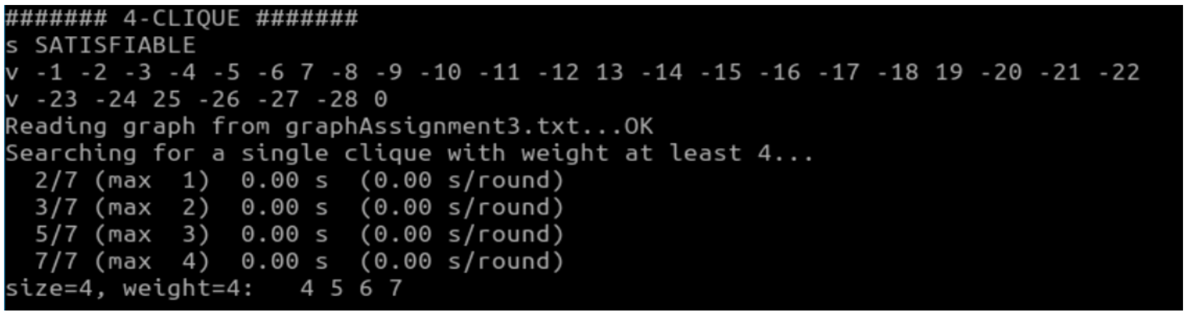
2.5 Main

The main function calls all the functions mentioned before to test our algorithms. We have created the graph included in the Assignment guide in DIMACS format (txt file). In this function, we create a graph out of this file, and start calling *Picosat*, *Cliquer*, and the search and optimization solver functions.

3 Results

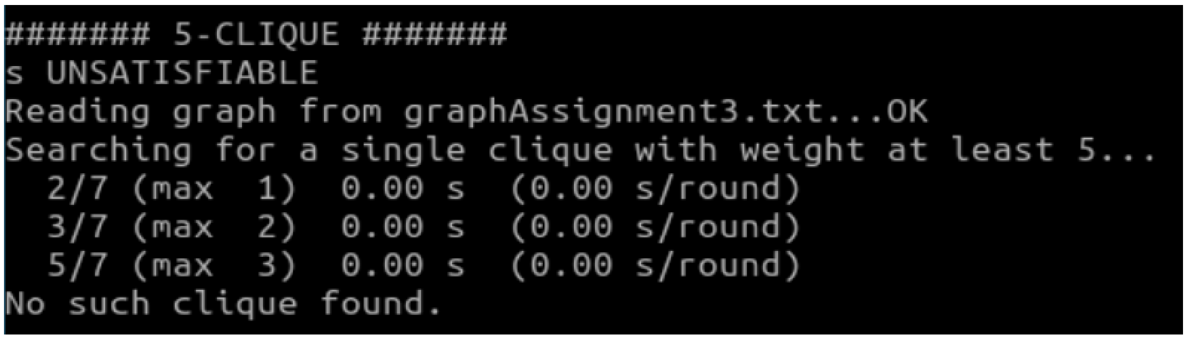
First, we are going to solve the decision version of both problems with *Picosat* and *Cliquer*. We want these programs to return if a solution exists or not, but in this case, the programs return the solutions for the search problem. As we can see in Figure 1, we the programs find a solution for both problems. For the CNF-SAT problem, the solution we get should be decoded by getting all the positive numbers, and calculating i and r (Recall the variables were $r * n + i$). However, we are not interested in the solution, just in its existence.

In Figure 2, we are able to see that the CNF-SAT instance is unsatisfiable and that there are no cliques of size 5 in the original graph.



```
####### 4-CLIQUE #######
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
v -23 -24 25 -26 -27 -28 0
Reading graph from graphAssignment3.txt...OK
Searching for a single clique with weight at least 4...
  2/7 (max  1)  0.00 s  (0.00 s/round)
  3/7 (max  2)  0.00 s  (0.00 s/round)
  5/7 (max  3)  0.00 s  (0.00 s/round)
  7/7 (max  4)  0.00 s  (0.00 s/round)
size=4, weight=4:   4 5 6 7
```

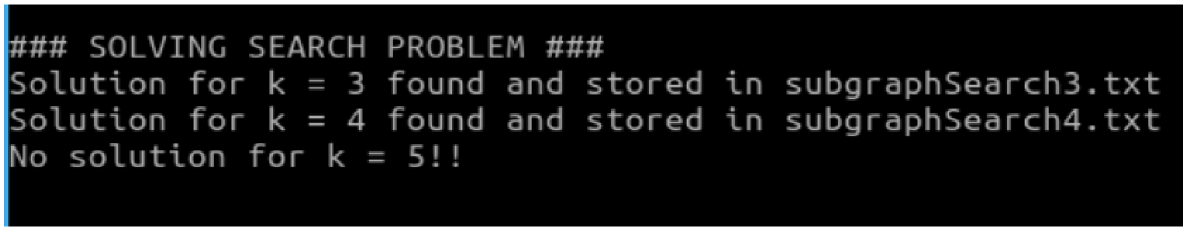
Figure 1: Deciding if the graph has a 4-clique.



```
####### 5-CLIQUE #######
s UNSATISFIABLE
Reading graph from graphAssignment3.txt...OK
Searching for a single clique with weight at least 5...
  2/7 (max  1)  0.00 s  (0.00 s/round)
  3/7 (max  2)  0.00 s  (0.00 s/round)
  5/7 (max  3)  0.00 s  (0.00 s/round)
No such clique found.
```

Figure 2: Deciding if the graph has a 5-clique.

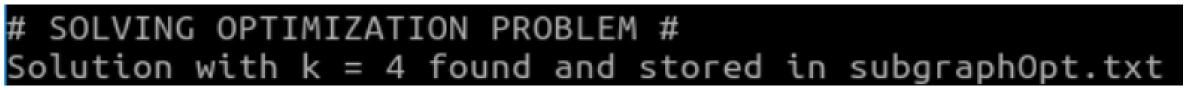
After that, we compute the search and decision versions of the clique problem by making queries to our oracle *Picosat*, which solves the decision version of the CNF-SAT problem. Figure 3 shows that we were able to find the k-clique of this graph for $k < 5$, but we could not find a 5-clique. If we found a solution, we stored it in a txt file.



```
### SOLVING SEARCH PROBLEM ###
Solution for k = 3 found and stored in subgraphSearch3.txt
Solution for k = 4 found and stored in subgraphSearch4.txt
No solution for k = 5!!
```

Figure 3: Search problem output, $3 \leq k \leq 5$.

Finally, we compute the optimization problem of the clique problem by using the decision version of the CNF-SAT problem (see Figure 4). In this case, we always find a solution, as we are assuming that the graph has at least one vertex. The solution is also stored in a txt file.



```
# SOLVING OPTIMIZATION PROBLEM #
Solution with k = 4 found and stored in subgraphOpt.txt
```

Figure 4: Optimization problem output.

The txt files are created with our Writer. It will store a subgraph such that all the vertexes in the clique are connected, and the rest of the vertexes are isolated.

4 Conclusions

In this assignment, we solved the clique problem by reducing it to a CNF-SAT problem. We were able to find out that if we solved the CNF-SAT decision problem with the reduced instance of our original problem (clique decision problem), we could solve the latter. This is because this is a Karp-reduction. However, when we solved the CNF-SAT problem in exercises 5 and 6, *Picosat* returned the solution for the instance, although we were not interested in the solutions for those exercises (we were focused on the decision version of the problems). If we wanted the solutions for the clique search problem, we would have to translate the results. In that case, we would be talking about a Levin-reduction.

Furthermore, we have implemented two algorithms to solve the search version and the optimization version of the clique problem by using the decision version of the CNF-SAT problem as an oracle. We have proved that we could get a solution just by removing a vertex and checking if the subgraph had a k-clique or not. For the optimization problem, we have used a similar idea to the one used in the reference book of this course, in the proof that the maximization problem related to a search problem was reducible to the exceeding a threshold problem. As the space of values is small enough, we can efficiently get the optimal value, and then we find the solution whose value is the greatest.

5 Questions

1. *What kind of reduction is obtained?*
   If we are talking about the decision version of these problems, this is a Karp-reduction, as we are only mapping the input to an instance of the reduced problem (CNF). *Picosat* determines whether the formula is satisfiable or not, and the result will be the same for the clique problem. We have done exercises 5 and 6 so that *Picosat* solves the search problem, and this result should be translated. In this case, we could say that the search version of these problems leads to a Levin-reduction.

2. *How many clauses are generated in terms of n and k?*

   - P1: k clauses.
   - P2: $n * \frac{k*(k-1)}{2}$
   - P3: $k * (k-1) * |\overline{E}|$, where $|\overline{E}|$ is the number of pairs of vertexes that are not connected.