ESCOLA
POLITÈCNICA SUPERIOR
UNIVERSITAT DE LLEIDA
INSPIRING THE FUTURE

Intel·ligència Artificial
Pràctica 2

por
Sebastian Jitaru i Gerard Llubes

Grau en Enginieria Informàtica
Intel·ligència Artificial
Universitat de Lleida
December 13, 2022

Contents

1 Desing decision **2**
1.1 graph.py 2
1.1.1 min_vertex_cover 2
1.1.2 max_clique 2
1.1.3 max_cut 2
1.1.4 13_wcnf 3
1.2 auct_solver.py 3

1 Desing decision

In this report we are going to talk about the decisions taken during the implementation of the algorithms. To make it more organised we are going to talk about the changes we made file by file.

1.1 graph.py

in the graph representation class we modified three functions. These being "min_vertex_cover", "max_clique" and "max_cut"

1.1.1 min_vertex_cover

to implement this algorithm we followed the contents of the slides. This algorithm returns a list of vertex, these being the essential ones to cover all the edges in the graph.
in the implementation we first create the nodes of the graph. Once the nodes are created we add the soft clauses, in this case these will be all the nodes in the graph and they will be given 1 as weight following, we add the hard clauses, in this case all the edges in the graph. because we don't define a weight for the edges it will be considered as infinite. then using the solver we can return the minimum nodes needed to cover all edges.

1.1.2 max_clique

to implement this algorithm we followed the contents given in the slides. This algorithm returns a list of nodes representing the vertex in a graph in which all nodes are fully connected to each other (complete graph).
first we create the nodes, once we have created the nodes we add the soft and hard clauses, the soft clauses being the vertex of the graph with weight 1 and the hard clauses being the edges that "are missing" in the graph, these being the edges left for the graph to be complete. We don't specify a weight to the hard clauses so that it is considered infinite.
then using the solver we return a list of nodes, these being the vertex in the graph that form the maximum clique.

1.1.3 max_cut

to implement this algorithm we followed the contents given in the slides.

1.1.4 13_wcnf

in this function we transform the wcnf formula to a 1-3 wcnf formula, soft clauses stay the same but for hard clauses if the clause is bigger than 3 we divide it in multiple clauses of maximum length of 3 using variables to connect the clauses by putting on one clause the varaible as it is and in the following clause the varaible falsified so that the clause as a whole stays the same.
we used an auxiliar function to make the division.

1.2 auct_solver.py

We implemented this class to handle the auction simulation.

```
class Auction:
    def __init__(self):
        self.bids = dict()
        self.agents = []
        self.formula = wcnf.WCNFFormula()
        self.goods = []
        self.args = parse_command_line_arguments()
        self.solver = msat_runner.MaxSATRunner(self.args.solver)

        self.readfile()
        self.SoftClauses()
        self.HardClauses()
        if self.args.no_min_win_bids:
            self.noMinWinBid()
        self.printResult()
```

Figure 1: DFS implementation code snippet

we've made a dictionary for bids, where the key is the bid number and the value is the bid itself, we also use the key as the soft clause for our implementation. We chose this data structure because we think is one of the most efficient data structure and made the implementation easier for us. For the rest of variables we used lists.
we put the "self.functionName()" in init so that they run when the class is instantiated.
the readfile function parses the data of the given file.
the softCauses function gets all the soft clauses according to the documentation presented in class.
the hardCauses function gets all the hard clauses according to the documentation presented in class.

the no_min_win_bid functions adds a soft clause for each agent containing the key of the bits where they participated