UNIVERSITAT DE LLEIDA

Escola Politècnica Superior

Grau en Enginyeria Informàtica

Aprenentatge i Raounament Automàtic

Primer parcial

Joaquim Picó Mora

PraLab2

Professorat : R.Bejar

Data : Divendres 17 d'abril

Contents

1 Introduction

This document will explain the strategies has been used in order to make a Local Search SAT Solver. Firstly, to take an idea of how SAT should work we organised the project in different modules and classes. When we saw more or less how SAT should perfom we start implementing the SAT in a single file in order to make it faster and don't waste time importing modules. Also, it was attempted to use lazy data structures in order to have an efficient experimental cost. In the next part, it will be explained the different strategies that we studied and found so as to choose the one that will compete in the SAT race.

2 Studied Strategies

The studied strategies were Walk-SAT and GSAT. Even though they were useful for understanding how an incomplete SAT works, our curiosity helps us to find new ones.

Our first implementation was the Walk-Sat that we've seen in the class. As it was our first implementation we weren't able to compare it with any other, so we decided to implement gsat in porpouse to see wich one of the two performs better.

After this we seen that our implementation of WalkSAT was the way better than the GSAT one.

Once we already implemented the two types of SATS that we seen in class, we start searching for a strategy that performs better than this both, and we found one quite interesting.

This new one was a combination of the Walksat Random restarts strategy with the functionaity of the GSAT. And that makes a lot of sense for us becouse we thought that GSAT probably was slower due to he stucs a lot more in local minimas than WalkSAT do. This strategy is called Random Walk GSAT.

Next in the graphics bellow we will show how the three strategies perform in a set of satisfiable and diferent sized formulas:

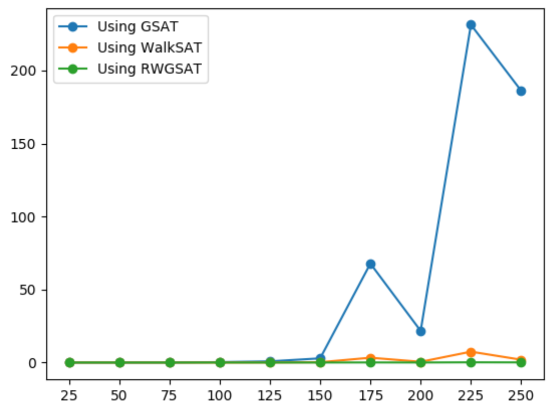

Figure 1: Graphic GSAT vs WalkSAT vs Random Walk GSAT

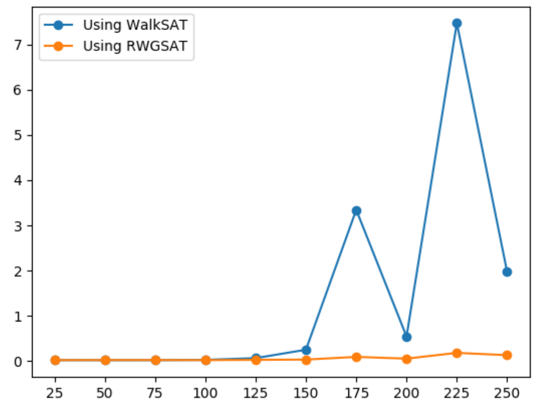

Figure 2: Graphic WalkSAT vs Random Walk GSAT

3 Chosen Strategies

3.1 GSAT with Random Walk

As the graphics above show, this strategy outperforms the other two, this is becouse it gets the best things of both others and put it toghether. Walksat skips better local minimas but it flips randomly the literal to get the next step. And GSAT gets stuked more time in local minimas but it choses better the literal to flip. So basically, this strategy skips a lot of local minimas doing hill climb moves and choses better the literal to flip (with the costs related of doing it).

Here we will show de pseudocode of the strategy:

```
for i in max_tries:
        for j in max_flips
                if (probability p):
                        pick a variable in some unsat clause
                                 and flip it in the interpretation
                else:
                        follow the standard GSAT scheme
                                -> Make the best possible move
```

3.1.1 Implementation details

4 Data Structures

In order to have efficient functions, there are some data structures that helps the solver to have better efficient functions. All of them are stored in *fracaSAT class.*

1. ***formula***. This data structure was given in class. It represents a list of literals in positive and negative that encapsulates a list of clauses that the literal appear. It is a fast way of getting all the clauses that appear a literal.

2. ***clauses***. A list of clauses with their literals. This data structure is needed to get the function cost of every interpretation.

3. ***not_found***. A list of literals that appear only in negative or positive. This data structure helps the solver to have a minimal set of clauses that evaluates true[1] and to get approach to local optimal. It is created when the program reads the CNF formula.

5 Statistics and Heuristics

In addition of developing different strategies in order to improving the solver performance, some heuristics have been created. These heuristics have been used in order to slightly modify the fracaSAT object. As a result, this heuristics will modify the behaviour of our normal strategy. The main aim of them are not to be again in the same local optimals modifying the cost of these interpretations. These are some of the thoughts heuristics and those which were developed.

1. **Creating a clause of the negate interpretation** not only will not only improve the cost of the search but also this one will be incremented, as the cost of having the cost of an interpretation will be harder. That is because half of the list of our structure will append a new clause.

2. **Repeating a clause** is a more interesting idea as it will not become as hard as the previous one and the length[2] of the new clause will be the same of the others clauses.

3. **Making a clause of the unsatisfied clauses** will prune better than the previous one despite the fact that the cost of getting the cost of the interpretation will be harder.

Even thought it is a good way to not lead to the same local minimal, it is very complicated to not overload too much the data structures with naive information.

[1]Only if the negate literal appears in the cnf formula.

[2]Number of literals

6 Conclusion

After the race we realized that a really important fact for the optimitzation of our sat were the values of max-flips and probability. So we decided to study of what value showld we assign to Max-flips, and we discovered that was the relation:

$$num\, flips = \frac{num\, clauses}{num\, lits} * 1000$$

This change makes our SAT a lot faster.

After all, even though we dont had a good place in the race, at least we had implemented a strategy that seems that works better than the classical ones.