Efficiency of SAT solvers

Georges Aaron RANDRIANAINA et Lauric DESAUW

January 6, 2020

1 Introduction

In the first part of the project we implemented some SAT solver with sat4j. We wanted to compare our solver to sate-of-the-art solvers to see how far we were from them. We wanted to compare to the following solvers: cryptominisat and minisat. Unfortunatly we didn't suceed to install minisat on our machines so we only compared to cryptominisat. To run the script that provide the database for the tests you have to redirect the output of eclipse in a file *result.csv* and then run the following command *python launch.py*. It will write a xtend files with unit test method (JUnit). Then you have to run the test of *BenchmarkSolver.xtend* to write the results in the file in the form of a csv file with the form *Benchmark/solver/version/sat/time (ms)* In the next of this report we will run some test to compare the efficiency of the solvers and their correctness. Then we will use those solvers to determine hard formula.

2 Best Solver

2.1 Experimental protocole

We have a set of SAT solver that use different method. Because we want to solve formula as fast as possible while having the correct awser we want to have a classement of those solvers to know which one we may use. To make our classement we are using the following protocole:

- For each solver we use the data set of formula: samplingfm
- We mesure how long they take to solves all the formula
- We check if all their awsers are corrects. If a solver have wrong anwsers he will not be considered in the classement.
- Then we do a classment of the mean of time taken to solve a formula. The most efficient solver is the one which's taking the less time.

2.2 Results

We use the results computed whith the launch.py script. We computed the mean solve time in *ms* for each solver.

solver	sat-4j-java	sat-4j-maven	sat-4j-jar	cryptominisat 5.6.7	cryptominisat 5.6.8
time (ms)	78 855.49	85 983.56	65 674.83	17 643.21	17 735.91

First we remark that the sat4j solver are far less efficient than the cryptominisat one. For the sat4j solver the jar is slighly better than the other sat4j solvers. We can explain that by the fact that the sat4j-maven have to build a project each time so it take some times. The new version of cryptominisat is a bit worse than the old one, because the difference is less than a percent that can be explained by a mere random variation.

3 Presence of Bugs

3.1 Experimental Protocole

In the previous experiment we've not consider the solvers that gave a wrong awser because we wanted to always have a correct awnser. But maybe some of those solvers are more efficient and have just a bug in some corner cases. First we want to know if the solvers are consistent ie they always gave the same awser for a formula. Next we want to know if they gave the correct awnser. We used the followings protocoles:

Consistent: For each solvers we are trying to know if they are consistent, if they always give the same responce for a given formula. For that we basicly run the solver several time on the same formula and watch if it always return the same.

Correct: For each solvers we want to know if it gaves the right anwser. To know the correct awnser we should use a proofed solver but it may take a lot of time. We rather compute the awnser of all the other solver and assume that the majority is right. This solution is natural because most of those solvers have been tested a lot and designed separatly so the probability that they all gave the wrong awsnser on the same formula is really low.

3.2 Results

For the test of consistency we obtain that all the solvers are consistent. One reason may be that we did not make enought repetition to find a bug. Another reason may be that some solver are deterministic and so are consistent and that non-deterministic solver are strong enought to be consistent (the alea may be change the solve time but not the awnser).

For the test of corectness we have that all the solver that gave an awnser all gave the same awnser. So we may conclude that our solvers are corret in most of the cases. We have to be careful because our solvers may not give the right awnser in some corner cases which are not in our dataset.

4 Harder SAT formula

4.1 Experimental Protocole

Up to here we have consider formula as an homogeneous set but some formula are harder than other to solve. It may be interesting to know which formula are harder and how solver react to them. We consider a formula harder for a solver if it solve time is higher than his mean solve time. Here, because we only want some formula and not all the hard one, we only focus on the hardest formula for each solver and watch if it his considered hard for the other solvers. We use the following protocole:

- We find the formula that take the most time for each solver.
- For each of those formula we look for all the solver if it is considered hard.
- We keep only the formulas that are hard for a majority of solver.

4.2 Results

We find that all the solver have the same hardest formula which is listReverse.sk_11_43.cnf. It mean solve time is 142 s. For these formula the most efficient solver is sat4j-jar and he is two time faster than the both version of cryptominisat. The sat4j-maven and sat4j-jave do not suceed to solve this formula.

5 Discussion

Our test are not hundred purcent accurate because we did not run them with an empty cache, RAM and a fresh machine. Moreover if we want to have a higher accuracy we have to use a bigger dataset.

Therefore, even with our dataset, we can conclude that all the solvers are correct and that the cryptominisat are more efficient that the sat4j solvers. But the sat4j solvers are more efficient for hard formula such as listReverse.sk_11_43.cnf.