COL876 - Project 1

JAHNABI ROY

10^{th} October, 2024

1 Project Overview

1.1 Implementation

A basic CDCL solver has been implemented which utilises VSIDS for ordering variables until one point and then switches to Random Choice Ordering after a certain number of conflicts have been reached. Backjumping, which is crucial to CDCL Solvers is implemented by finding the UIP nodes, learning the new clause and then restarting again from level 0. This has been implemented to be keep logic simple and also trying to prevent getting stuck in the same local minima.

1.1.1 self.F

This contains the CNF formula. New clauses learnt are added to the formula and the function is called on this revised CNF formula again.

1.1.2 self.model

This contains the model learnt from the current run of the CDCL solver. The dictionary is a list indexed by the variable, having its polarity, its status and its level assigned. The status of the variable assigned is according to how the decision was made - if it is a decision node, "DONE" is assigned and if it was a forced decision, then "FORCED" is assigned. Before any such assignment, the model is initialised with "U" and then after initial assignment, "A" is assigned.

1.1.3 Implication Graph

The Implication Nodes are made to store the variable, the polarity of the variable and the level in which decision is made. This is crucial for future steps of resolution and backjumping. The Implication Graph is implemented as an adjacency list, indexed by the level as their key and the nodes in that particular level in the list. This is crucial in understanding how the decisions have been made, and assigning predecessors and successors accordingly. The successors list is crucial in identifying the learnt clause. The predecessors list helps in backtracking and getting the relevant literals for the learnt clause.

1.1.4 Variable Ordering

VSIDS (Variable State Independent Decaying Sum) is primarily implemented for choosing how the variables are chosen for decision making; on basis of the occurrence of the literals, the corresponding literals have been assigned. After a particular number of iterations (finetuned by experiments), the solver switches to Random Variable Ordering to escape repeated states being added into the formula.

1.1.5 Backjumping

UIP nodes are identified with the help of the direct conflict clause. The first and last UIP nodes are identified depending on the existence of the learnt clause already in the formula or if the learnt clause from first UIP or last UIP is empty. If learnt clause from both UIPs are empty, then the solver concludes the formula is UNSAT and returns no model. Backjumping is always done to level 0, reinitialising model polarities and levels. This is done for ease of implementation. It also reduces chances of being stuck in the same loop of local minima; however not by much since repeated learnt clauses ultimately lead to similar repeating loop.

Another heuristic that is used here for learning clause is that if we get a repeated learnt clause then the CDCL solver shifts from learning the last UIP clause to the first UIP clause for a fixed number of iterations (fixed to five here). Then it goes back to learning the last UIP again.

1.1.6 Limit

The code has been limited to run only for 10k conflicts. This number has been decided for the sake of simplicity of trial checking on personal computer. After reaching this limit, the solver outputs UNSAT (LIMIT EXCEEDED), specifying the reason thus.

2 Testing

The tests folder contains tests : test_0 to test_28 – ranging from r= 0.2 to r=5.8. The check_assignment.py file was used to check if the assignment given by the SAT solver was indeed correct. PySAT was used to get the answer if a CNF formula was indeed SAT or UNSAT. (Crytominisat could not be set up in my device; thus the workaround).

3 Observations

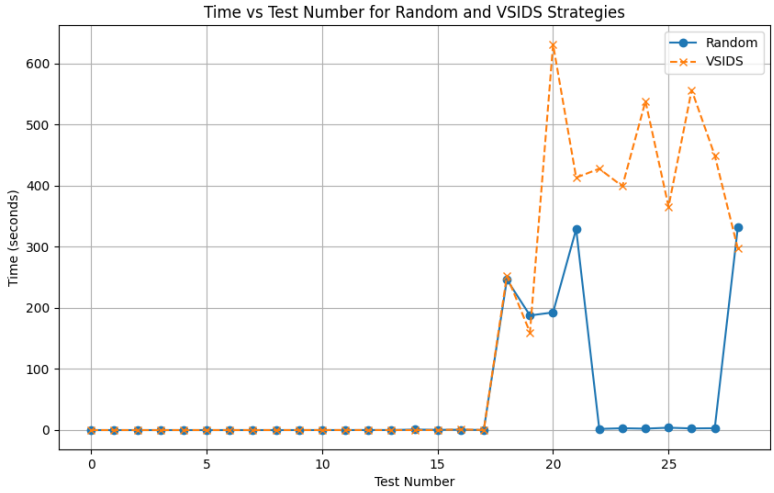
3.0.1 Random Choice vs VSIDS (in combination with random)

A general trend has been observed with both VSIDS and Random, upto r = 3.6, the solver works very fast, solving and return model/ output within 1 second. However from $r = 3.8$ to $r = 5.6$, it takes a long time to run all the iterations. But after 4.4, it is generally quite fast (within 100 seconds) in getting the answer.

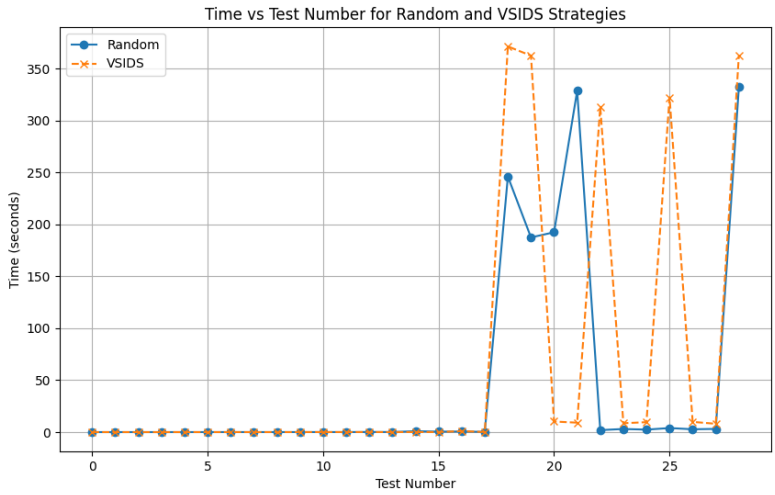
My comparison with Random won't be of very useful since my solver shifts to Random after a particular number of iterations; so depending on the conflicts, it is possible Random dominates the time and thus the plot might as well be similar.

The following is the plots obtained of random and VSIDS (where 10k is the limit on the maximum number of iterations solver can go through).

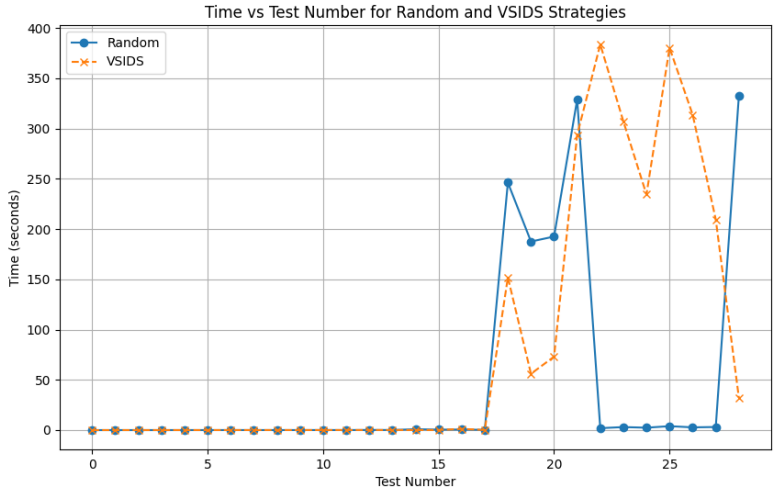
The below graph is the implementation with the 10k limit and 5000 as the number of conflicts after which VSIDS switches to random.



The below graph is the implementation with the 10k limit and 500 as the number of conflicts after which VSIDS switches to random.

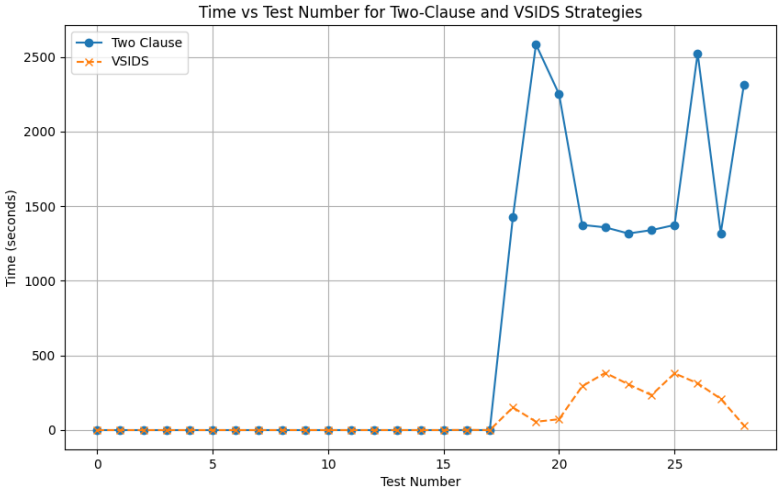


The below graph is the implementation with the 10k limit and 2000 as the number of conflicts after which VSIDS switches to random.



Observing from the above graph, **2000** was chosen as the number of conflicts after which VSIDS is replaced by Random Choice for picking the branching variable.

3.0.2 2-Clause Heuristic vs VSIDS (in combination with Random)



The above graph compares the current SAT solver implementation (2000 conflicts - shifting to Random) with Two Clause Heuristic. The results show significant improvement for VSIDS (in combination with Random) than Two Clause.

4 Other Observations and Improvements

With the heuristics chosen and how UNSAT condition is implemented, there is room for wrong answers as well. Getting an empty learnt clause is also marked as UNSAT when in fact it could be accounted to the fact that there might be a room for more exploration which is stopped due to no new learning in the same minima. This is an experimental observation and is not backed by proof however, it possibly explains the case observed for tests 18-21 (r=3.8 to r=4.4).

In most of the cases from and beyond test case 18 (r=3.8), it is observed that time limit exceeded is the primary cause of the solver being unable to return a valid model if SAT or simply return UNSAT. One particular observation to this is the fact that repeated clauses being added is a primary reason for limit to be exceeded. Thus, the thought being shifting to random ordering of variables. From the above graph, it is evident that random ordering dominates for test_22 to test_27. This observation motivates the use of random ordering in the CDCL Solver Implementation.

Since we always back jump to level 0, we should be concerned with the fact that we are probably stuck in the same loop of conflict due to the same clauses being learnt over and over again. But one improvement which should have been implemented is restarting the entire solver after a particular condition. How-

ever, given the structure of the code, restarting the entire model with VSIDS would not have led to new learning possibly due to the deterministic modelling of how variables are chosen within the CDCL solver and how UIP nodes are chosen as well. So, it was opted out from implementation.

Other than the above mentioned heuristics, clause removal was an important heuristic which was not implemented in the current CDCL solver. This would have allowed for only important clauses to have been kept in the formula while, allowing to escape the loop of learning repeated clauses. However the time taken could have increased significantly and possibly might require the number of conflicts to be more than what is currently to completely explore search space. So, for the current basic implementation, it was not opted for.

The idea behind the backjumping heuristic of oscillating between choosing the first UIP and the last UIP node is to escape any chance of getting stuck in the same exploration space and escape the local minima. The number chosen is fixed by experimentation.