Assignment Report
SAT SOLVER (SATisPy)

Snehal Raj (170705)
Keshav Bansal(170335)

Abstract-We were supposed to implement a SAT solver to to check whether a given DIMACS encoding SATISFIABLE or UNSATISFIABLE and provide a satisfiable interpretation in the former case.

Heuristic1-:

In this we applied Unit Propogation assigning deterministic values to each of the unit clauses formed as we progress in the semantic tableau.

As in case of most SAT solvers, our SAT solver too spends around 80% of time in unit propogation(verified experimentally).

Method of selecting literal **:**While experimenting we found out that when each literal is broken away from the clause one by one, it allows for maximum unit propagation along the way.

Method of selecting clause: We tried various methods such as sorting the list of clauses in both ascending and descending order according to the length of the clause .Though it worked well in some cases, solving time was lesser if we worked on the clauses in the same order they were supplied(In some cases time was reduced upto 50%)

As we progress in the semantic tableau we maintain an array consisting of boolean values currenty assigned to the variables and using this any clause containing literal in same state is deleted while any literal of opposite state is removed from the clause.This heuristic is ideal in case when we are given some unit clauses to start since each step of propogation may lead to other unit literal being formed and thus as our algorithm progresses it becomes exponentially faster.

Though this works really fast for sudoku encodings(even comparable to MINISAT in some cases),it was not so apt for 3-SAT problems

Heuristic 2

The first heuristic works like a charm if there a lot of unit literals present in the encoding but while experimenting with different encodings we found out that when there are not many unit literals to begin with, our algorithm takes a lot of time to tackle that problem. Thus, to solve this issue, we implemented another heuristic for specifically those encodings where the number of unit literals are too few.

We start by sorting the literals on the basis of their decreasing order of frequency of occurence in the encoding. Then, we choose the first 20 literals from this sorted array and start guessing their valuations for which we may find our satisfiable solution. We permute over each possible interpretation of this set of 20 literals and then check if that particular interpretation can lead to a satisfiable solution or not

While experimenting we also observed that assuming all literals to be false and then giving them true values one by one is better than the other way (i.e. to guess all of them to be true and then go on assigning false values one by one). As we have assigned truth values to 20 literals, our encoding now works as if there were 20 unit literals in the beginning. This improves the time for checking for the existence of a satisfiable interpretation drastically. While running over all permutations of truth values for those 20 literals, we break the loop as soon as we find a satisfiable interpretation. And that serves as a solution for our problem.

Heuristic 3:

If a propositional variable occurs with only one polarity in the formula, it is called *pure*. Pure literals can always be assigned in a way that makes all clauses containing them true. Thus, these clauses do not constrain the search anymore and can be deleted.This heuristic did not work as expected and solving time was increased making us reject it.SInce it is currently used is many SAT solvers we feel that it requires the use of some special data structure.

While experimenting with some of the heuristics for the sat solver we also came across some heuristics which were not as good and as a result we had to reject them.

Frequency based Sorting: We tried to incorporate sorting to make our algorithm more efficient.

We implemented sorting the complete array of clauses on the basis of the length of each close first in ascending order and then in descending. We also employed sorting clauses in an increasing order but then selecting clauses for decomposition based on decreasing order of frequency and vice versa. But unfortunately, none of these heuristics gave any good enough results to incorporate them in our final implementation of the SAT solver.

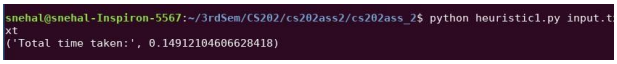
Implementation of Self-defined copy function: We noticed that our SAT solver was taking a little too much time initially so we started checking the time consumed in different sections of our program and observed that a lot of time of time was being used in copying and restructuring the arrays using the predefined functions of numpy. So we chose to use lists of lists instead of numpy arrays.

In place of the default delete functions of numpy, we employed the default delete function of lists and in place of the copy function of numpy, we started experimenting with different methods of copying arrays (deepcopy, cloning etc). At last, we ended up writing self-defined copy functions using slicing as that turned out to be a lot faster than other alternatives.

This resulted in a significant improvement in the times taken by our SAT solver and thus the same copying functions have been employed in both of the heuristics. Moreover, as copying consumes a lot of time, we refactored our entire code so that the need for copying arrays is minimized.

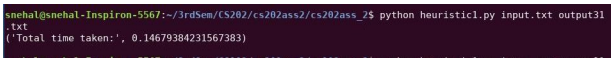
Outputs for some of the test cases:

- *Minimal sudoku puzzle generated in first assignment*



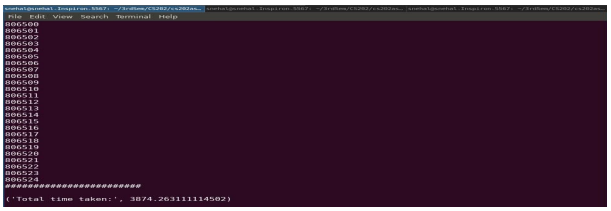
TIme:0.28 s

- *UF-022:*



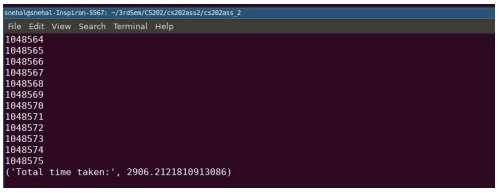
Time:0.1467 s

- *UF-033*



TIme:3874.26 s

- *UUF-021*



TIme: 2906.2121 s