COS 516 Final Project
Testing SAT Heuristics

Natalia Perina, Pranjit Kalita

January 17 2016

1 Introduction

Motivation: Learn how minisat SAT solver decision heuristics affect solver performance on benchmark instances.

Problem Statement: Use an existing trace generator and trace analyzer to measure minisat performance with varying decision heuristics (VSIDS vs. Random), and formula preprocessing techniques (VSIDS on/off) on benchmark problems. Also formulate the Hamiltonian Cycles K-Coloring problems on minisat, and test for both SAT and UNSAT instances on each solver.

2 Overview

A major portion of the project involved learning to utilize the trace generator and trace analyzer, with each subsequent stage outlined in Figure 1.

Secondly, in the beginning of the project, in order to get more acquainted with minisat, we modeled the Traveling Salesman problem, which was used with the previously modeled k-Coloring graph problem to test for relatively small difference with different heuristic selection, as well as preprocessing techniques.

Finally, the third phase of the project involved running on various SAT benchmark files minisat with random decision heuristic and VSIDS with proprocessing turned off.

3 Project Tasks

There were three main parts to the project -

- Modeling some famous NP-hard problems, namely k-coloring, largest common subset, and traveling salesman
- Comparing the effects of formula preprocessing on SAT benchmark files
- Comparing the effects of random decision heuristic on SAT benchmark files

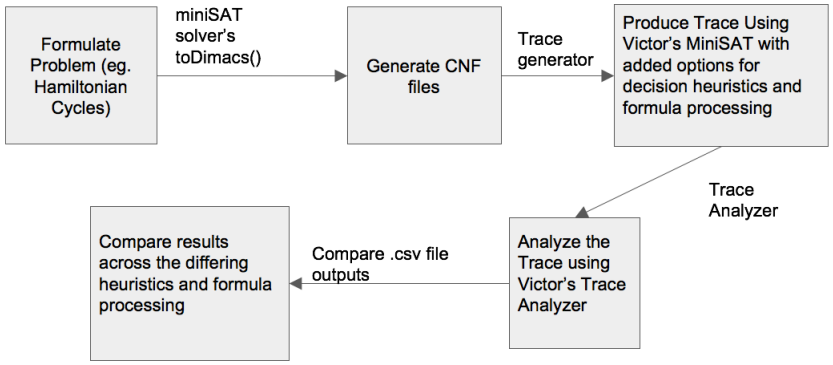

Figure 1: General overview of trace generator and trace analyzer tool usage

We used minisat and Victor Ying's trace generator and trace analyzer to compute the effects of changing heuristics through outputs generated via those tools. A major amount of time was spent to understand the structure of these outputs, and how they corroborated with changing decisions.

Natalia helped Pranjit with writing out some of the important clauses for the traveling salesman problem. She had earlier unsuccessfully tried to model the largest common subset problem. She also co-worked with Pranjit on testing out different smaller files with varying percentages of random variable selection. Finally, she worked on VSIDS formula preprocessing turned off and compared their results with the default VSIDS preprocessing on results on large SAT benchmark files.

Pranjit developed the modeling for Traveling Salesman problem and his k-Coloring modeling was used for testing earlier. He also co-worked with Natalia on testing out different smaller files with varying percentages of random variable selection. He worked unsuccessfully to alter the code inside pickBranchLiteral() to make the process of randomizing more predictable, eventually he had to rely on altering only random seed and random frequency values to achieve a high degree of randomness on benchmark SAT files. He also worked with testing the effects of random decision on benchmark files.

In order to test randomness and to make it more determinable, looking at the code of pickBranchLiteral() of minisat's solver, a certain if condition determines whether or not a random variable assignment is being made. In order to take the random branch, the following was tried -

- Put the if condition within a while loop so that random decision has to be made - Did not work since it kept going into an infinite loop (Figure 2).
- Change the random frequency and random seed values that the result of the if condition depended on these two values. We tried with various value for both and got 70% randomness (on smaller files in the beginning) with

a seed of 0.3 and frequency of 0.9. Upon sticking with the same values, for the larger benchmark files that we were able to check for, randomness values of upto 77% was achieved.

The Minisat SimpSolver comes with the option to turn on and off cnf pre-processing. Victor's tool, however, was built using the regular solver that does not have these capabilities. So to run cnf instances with pre-processing on, we first used the regular Minisat SimpSolver to pre-process our benchmark files and then output the pre-processed cnf instance to a dimacs file. We did this using the dimacs flag of the SimpSolver. We were then able to input pre-processed CNF files to Victor's minisat solver which could generate the traces for analysis.

4 Results

4.1 VSIDS Pre-Processing On

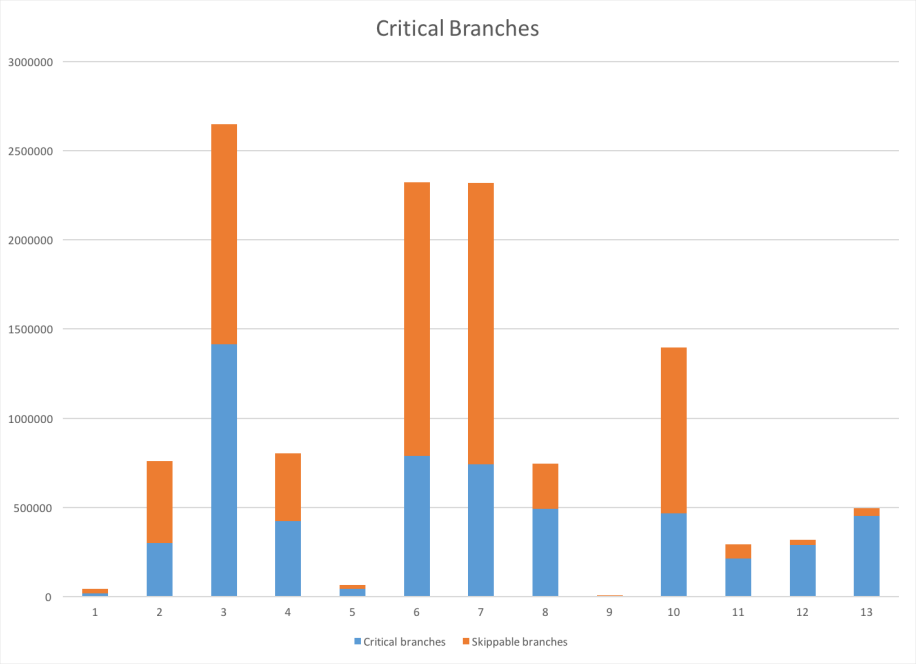

Figure 2: The number of branches that were required or avoidable. Results include 14 unsatisfiable instances from Victor's benchmark files that could be completed without running out of memory or storage.

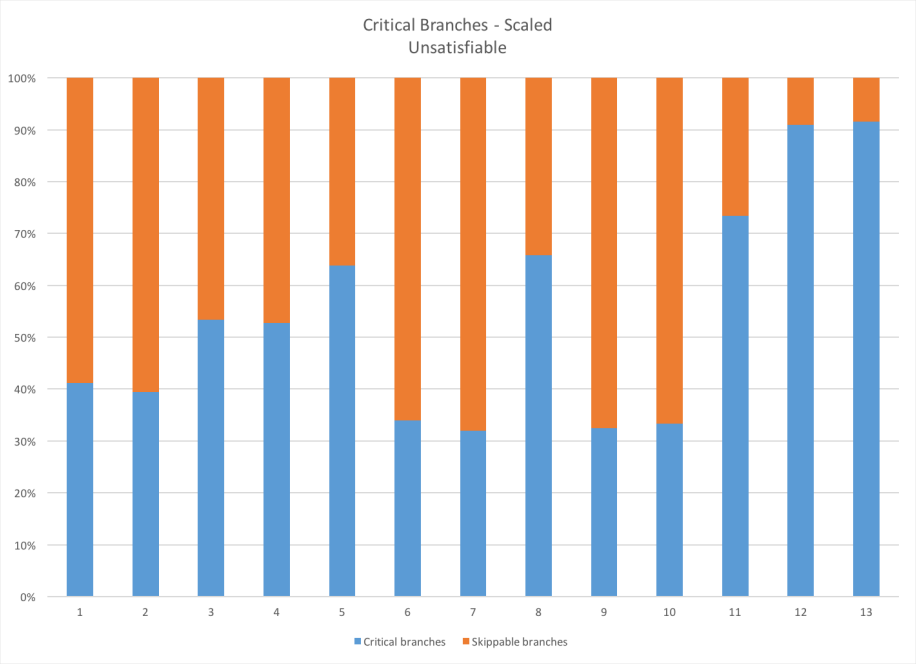

Figure 3: The number of branches that were required or wasted. Results include the 14 unsatisfiable instances from Victor's benchmark files that could be completed without running out of memory or storage.

Summing over all unsatisfiable instances, 46% of branches are required and 54% of branches are wasted. These results differed from Victor's which reported 63% required and 37% wasted.

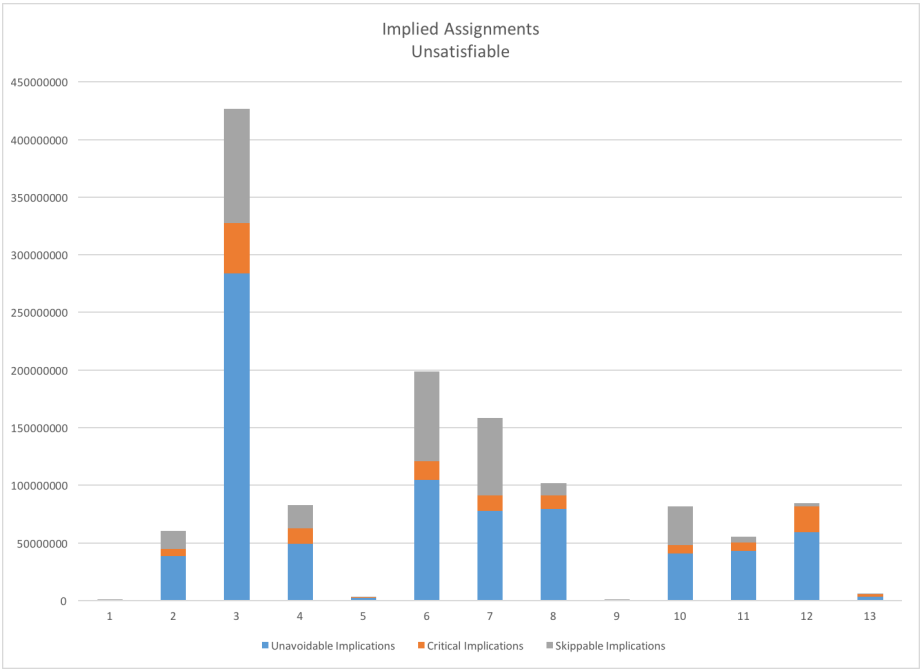

Figure 4: The number of branches that were required or wasted. Results include the 14 unsatisfiable instances from Victor's benchmark files that could be completed without running out of memory or storage.

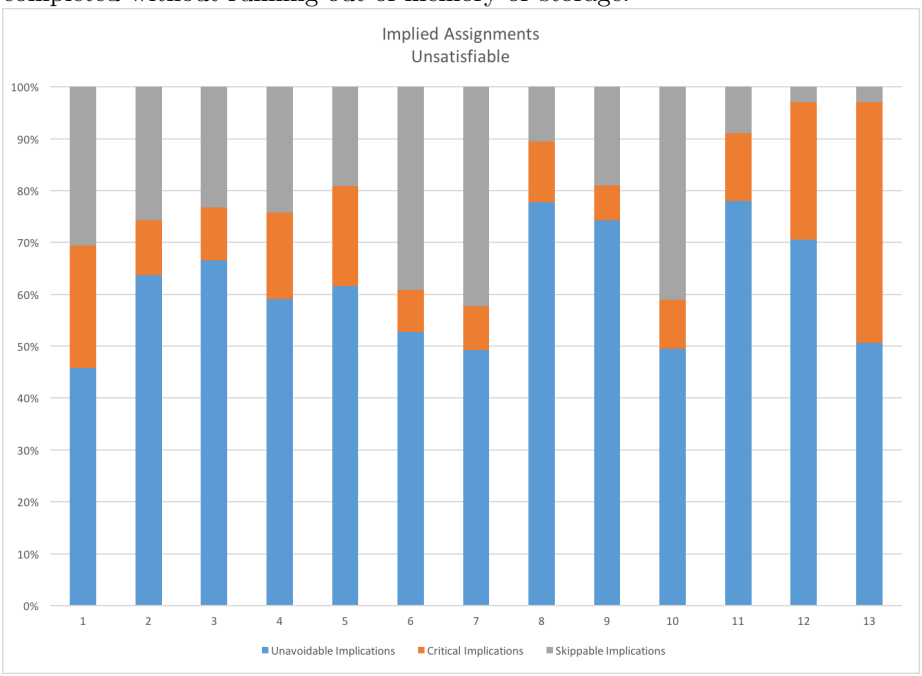

Figure 5: The number of branches that were required or wasted. Results include the 14 unsatisfiable instances from Victor's benchmark files that could be completed without running out of memory or storage.

Again summing over all unsatisfiable instances, we get that 12% of implications are required, 26% are wasted, and 62% are unavoidable. Here the percentage we obtained for required implications is approximately the same as the 13% reported by Victor. His results without pre-processing however differed in their distribution of wasted and unavoidable conflicts - he had a larger percentage of unavoidable implications at 74% and a lower percentage of wasted assignments at 13%.

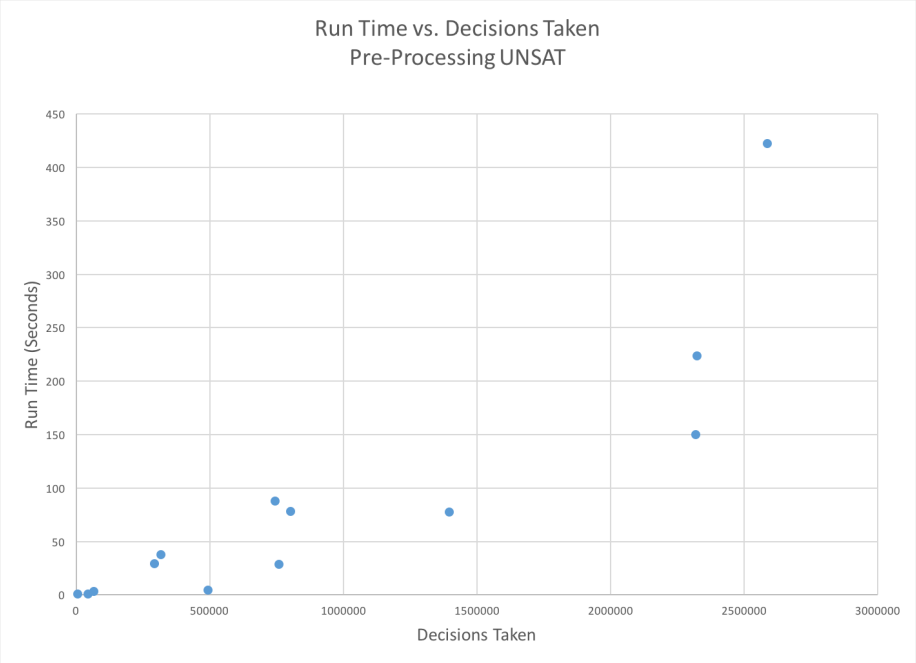

Figure 6: The run time plotted against the decisions taken for all 14 unsatisifiable pre-processed instances.

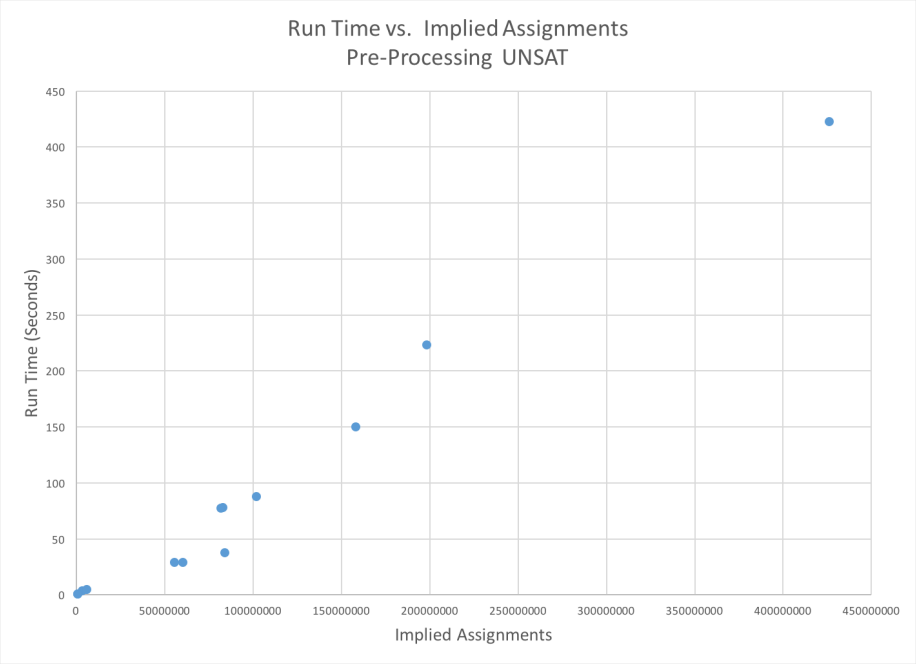

Figure 7: The number of implied assignments plotted against hte runtime for all 14 UNSAT instances.

In figure five we plot the run time against the number of decisions taken by the solver. These results coincide with Victor's results who points out that in other research it has been explored that the number of branches explored is a poor predictor of run time. Like Victor did as well, we see a stronger linear trend in Figure 6, the plot of run time against the implied assignments. This is consistent with Victor's findings, and as he points, out with "the understanding that most time is spent searching for implications in BCP."

4.2 Random

Unfortunately, we weren't able to accomplish the results we had hoped for with random decision heuristic on large benchmark files.

- Upon the first try at running Victor's trace analyzer on an UNSAT instance, the trace output exceeded more than 490 GB, thereby making it practically impossible for us to get any results.
- We think that either Victor's trace analyzer was not prepared to handle random decision heuristic, although we could not find out why as it was beyond the scopt of the project.
- Upon halting execution, we had a running randomness %age of 77.31%

(21060 restarts, 14194302 conflicts, 139989905 decisions, 36822210731 propagations, 96386684707 conflict literals).

- The extreme disk memory required coupled with the immensely long time taken (over 20 hours to get to the above point) made us abandon further experimenting, since we couldn't find a solution to it.
- Furthermore, upon trying the trace analyzer on the trace file from hlted execution, we got an assertion error, but we had ensured that the VSIDS version of the same cnf instance both gave us a trace (11 GB) and successfully ran on the trace analyzer.

5 Discussion

- The Pre-processing results allow us to read into how such a heuristic might be able to improve the minisat solver's performance.
  - Victor suggests in his thesis that, as we can see from figure 6, implied assignments seems to serve as a good predictor of solver runtime, then "we may use the proportions of wasted and required implied assignments as a proxy for the degree of improvement in solver run times."
  - So does pre-processing potentially improve the solver's performance? As noted previously, when summing across all instances, pre-processing analysis still presented with a total of 26% waste implications. This was more than what Victor had when he ran without pre-processing. This implies that it may not do much to improve solver performance on large instances.
- We could not get results of random decision heuristic due to the issues highlighted above.
- It is hard for us to conjecture why, but we believe before we know why, we need to answer the question of how the trace outputs produced for them could be so impossibly large.
- We weren't able to model the largest common subset NP-hard problem in minisat.

6 Conclusions and Future Work

6.1 VSIDS Pre Processing On

- There is definitely more work to be done in terms of learning about how pre-processing and other heuristics can be used to improve solver performance.

- Our results don't show a significant difference between our pre-processing-on results and Victor's pre-processing-off results. It may be that on large instances, it is less effective, although this does not seem intuitive.
- We came across several issues with memory management and solver performance when running our tests, and as such, our results are limited. To facilitate true robust testing of the solver, work on developing an analyzer that can produce the same results without crashing due to memory or storage bugs is necessary.

6.2 Random Decision

- No tangible results obtained.
- A deeper look should be taken into what caused the trace outputs to be so seemingly large and non-halting. This should be used as a project topic in itself we feel.
- Also, why did Victor's trace analyzer produce assertion errors on random decision trace outputs remains to be seen, and is worthy of a more academic look.
- Random decision effects thereby need to be looked at in more detail.

7 Acknowledgements and References

We would like to thank both Professor Gupta and Yakir Vizel for their guidance throughout this project. Furthermore, we would like to thank Victor Ying for allowing us to use his trace generator and trace analyzer tools to use in our project. Victor's source code for his tool can be found at - https://github.com/YingVictor/sat-analysis.