UNIVERSITY OF CALIFORNIA, BERKELEY

CS270 Final Project - MAX SAT Algorithms

Hugh Chen and Yiwen Song

May 12, 2016

1 ABSTRACT

Our research topics of interest are the various algorithms used to solve the maximum satisfiability (MAXSAT) problem. We started by implementing three algorithms for MAXSAT in order to explore derandomization as well as linear programming for approximate algorithms. The three algorithms we implemented at first were a 2/3 derandomization algorithm, a 3/4 expected value approximate algorithm, and a 3/4 deterministic algorithm. After analyzing the results of these algorithms, it became clear that in spite of their empirical analyses, the actual performance may vary. As a result we analyzed two additional algorithms: a uniform classifier and a neural network classifier for choosing a solver.

2 ALGORITHMS

2.1 OPTIMAL EXHAUSTIVE ($exhaustive\_sat$)

The first algorithm is an exhaustive algorithm that checks every possible assignment of variables. This was straightforward to implement.

2.2 (2/3) DERANDOMIZED MAX SAT ($greedy\_sat$)

The second algorithm is a derandomized algorithm based on the simplest randomized algorithm for MAX SAT. The randomized version of this algorithm is simply to take the satisfy each clause with probability $1/2$. Simply by linearity of expectations, it can be shown that this algorithm has expected value equal to $1/2W$, where W is the total weight of all clauses (for our project we assigned a value of 1 to all clauses.

The derandomized algorithm is actually just an application of the method of conditional expectation, which pretty much just iteratively assigns variables based on the expected value of the clauses after each assignment. This algorithm is known as Johnson's algorithm [3] and Chen, Friesen, and Zhang [2] actually showed that the approximation ratio of this derandomized algorithm is 2/3.

2.3 (3/4) Stochastic MAX SAT ($randlp\_sat$)

The third algorithm we implemented was actually one that we proved to be an expected 3/4 approximation to MAX SAT in homework three. The algorithm essentially took advantage of a linear program to compute a relaxed version of MAX SAT, where clauses can be partially satisfied. Then we use results of the LP to assign the variables using Bernoulli distributions. It turns out that doing the LP rounding with 1/2 probability versus uniform rounding with 1/2 has an expected performance of 3/4, which is a definitive improvement over the uniform stochastic algorithm that we mentioned. In this project, we're also interested in investigating the mean and the variance of this algorithm empirically, to see if it would even be feasible in practice.

2.4 (3/4) Deterministic MAX SAT ($detlp\_sat$)

The fourth algorithm was a deterministic 3/4 algorithm for MAX SAT developed by Anke van Zuylen [5]. This algorithm looks to be a combination of a potential function based heuristic for variables (from Poloczek and Schnitger [4]) and the linear programming technique we had earlier. To briefly sum it up, we calculate $\alpha = (W_i + F_i - \bar{W_i})/(F_i + \bar{F_i})$. W_i and $\bar{W_i}$ are the sums of the weight of unsatisfied clauses that contain x_i and $\bar{x}_i$ respectively, but do not contain $x_{i+1}, ..., x_n$. F_i and $\bar{F_i}$ are the sums of the weight of all remaining unsatisfied clauses that contain x_i and $\bar{x}_i$ respectively. Then we deterministically set x_i to 1 if $\alpha \geq 1$ or if $\alpha \in (0, 1)$ and $q_i^* \geq (1-\alpha)/2\alpha$, where q_i is the solution to the linear program we used in $randlp\_sat$, otherwise we set x_i to 0.

2.5 Average Solver MAX SAT ($average\_sat$)

The fifth algorithm was quite simple. We basically uniformly chose between our three non-exhaustive algorithms. This algorithm was inspired by the homework algorithm which averaged between the uniform solver and the random linear program. Another way to look at it is as a sort of preliminary ensemble algorithm. A related work is the actual ensemble version of MAX SAT solvers using random decision forests [1]. In this paper, they discuss feature extraction from CNF formulas as well their machine learning approach to this classification test.

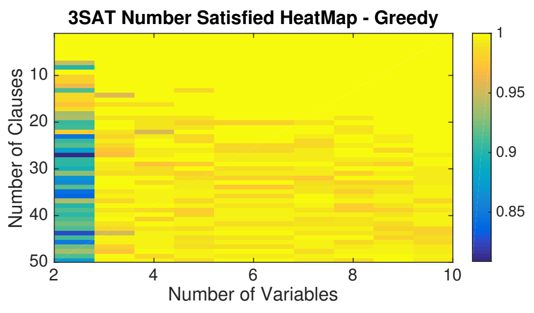
2.6 Neural Network MAX SAT (nn_sat)

This algorithm is another original algorithm that we attempt to use in this project, which extends upon the naive $average\_sat$ ensemble algorithm. The idea is that we observed that $greedy\_sat$ performs a great deal better than the other algorithms on the random inputs we used, we wanted to identify which cases were "hard" to $greedy\_sat$. To do this, we encoded features that were just the eigenvalues of two adjacency matrices. The first adjacency matrix used variables as vertices and clauses as edges. The second used clauses as vertices and variables as edges. We used eigenvalues in order to convey the connectivity of our CNF formulas. Ideally, the neural network would classify an input to be best solved by either $greedy\_sat$, $randlp\_sat$, and $detlp\_sat$. Ultimately, our neural network wasn't successful in differentiating inputs and it unilaterally chose $greedy\_sat$, we'll further discuss why in section 3.

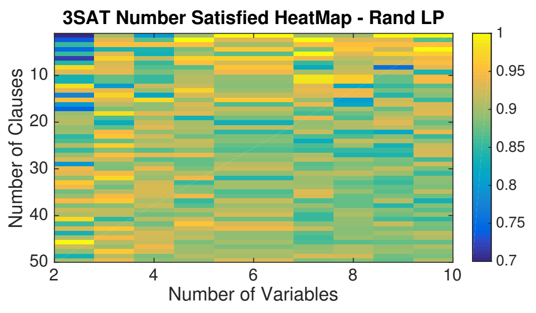
3 Conclusion

3.1 Results

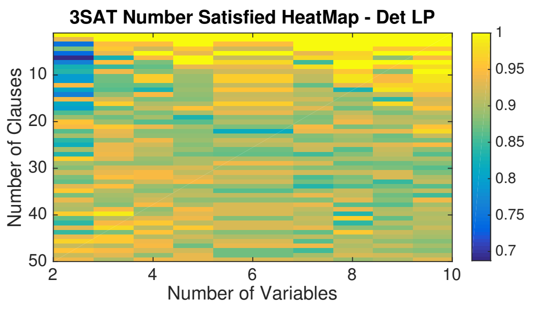
We generated random sets of data given three parameters: k (number of literals), m (number of clauses), n (number of variables). Generating uniform samples given these parameters becomes a fairly straightforward problem of sampling variables and flipping them randomly.



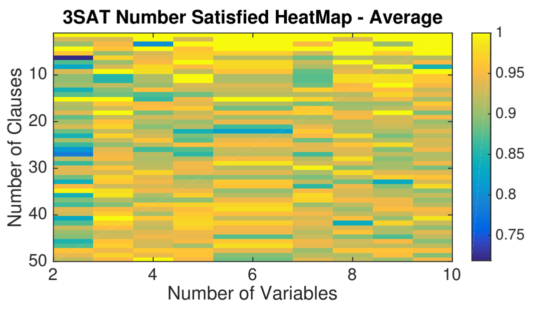
(a) $greedy\_sat$



(b) $randlp\_sat$



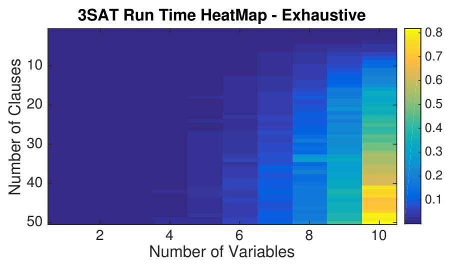
(c) $detlp\_sat$



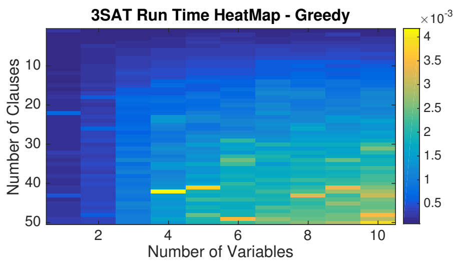
(d) $average\_sat$

Figure 3.1: We generated five random sets of data for 3SAT with number of variables ranging from 2 to 10 and number of clauses ranging from 1 to 50. Then we ran all of the algorithms across the data sets and found the average performance.

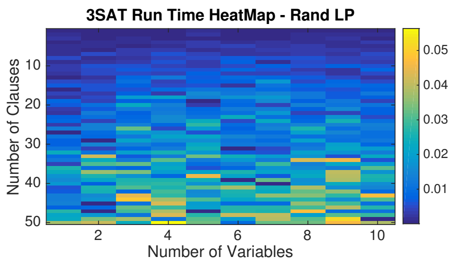
The first results we look at are across five random sets of data for 3SAT, with a range of cardinality for both variables and clauses. We plotted this in a heatmap to get a bit of a sense where different algorithms do better. This becomes the inspiration for our machine learning approach to choosing a SAT solver. We can see that even across the average of five sets of data, there are different areas we certain algorithms perform better. In figure 3.1 we can see a bit of the difference in performance. Additionally we recorded the run times across the algorithms in order to illustrate the differences in complexity (in figure 3.2). In certain cases it may be worth it to simply use algorithms with lower performance guarantees, because the performance is strong enough empirically and their complexity is low. Conversely, we also see in subfigure 3.2.a that the performance of the exhaustive solver heavily depends on the number of variables, in which case there may still be certain times when it's tractable to use.



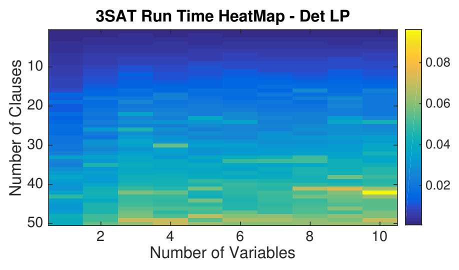
(a) *exhaustive_sat*



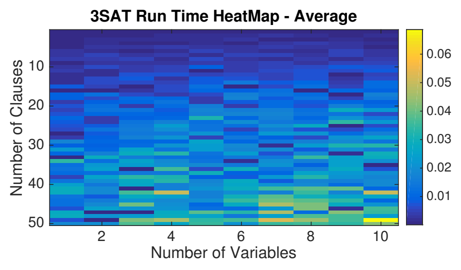
(b) *greedy_sat*



(c) *randlp_sat*



(d) *detlp_sat*



(e) *average_sat*

Figure 3.2: We generated five random sets of data for 3SAT with number of variables ranging from 2 to 10 and number of clauses ranging from 1 to 50. Then we ran all of the algorithms across the data sets and found the average runtime. The colors represent the ratio of the sat solvers' performance over the optimal performance.

In figure 3.2, we can also see that the *greedy_sat* may actually outperform the other algorithms quite handily. This is surprising because the greedy algorithm is actually the one with the lowest bound on it's performance in the proof. This unilateral performance inspired us to try to figure out what is "hard" to this conditionally derandomized algorithm. Unfortunately the algorithm performed so much better that we were either unable to generate enough data in order to give the neural net enough to go off of or we simply didn't have the resources to run an example with large enough parameters.

The second results that we look at are across different values of k, since all the previous example were on $k = 3$ examples with other parameters varying. For these varying k values we use a constant number of variables (10) and a varying number of clauses, in order to satisfy this critical clause-to-variables heuristic of $c = (log(2^k/(2^k-1)))^{-1}$. This critical heuristic was borrowed from: https://toughsat.appspot.com/.

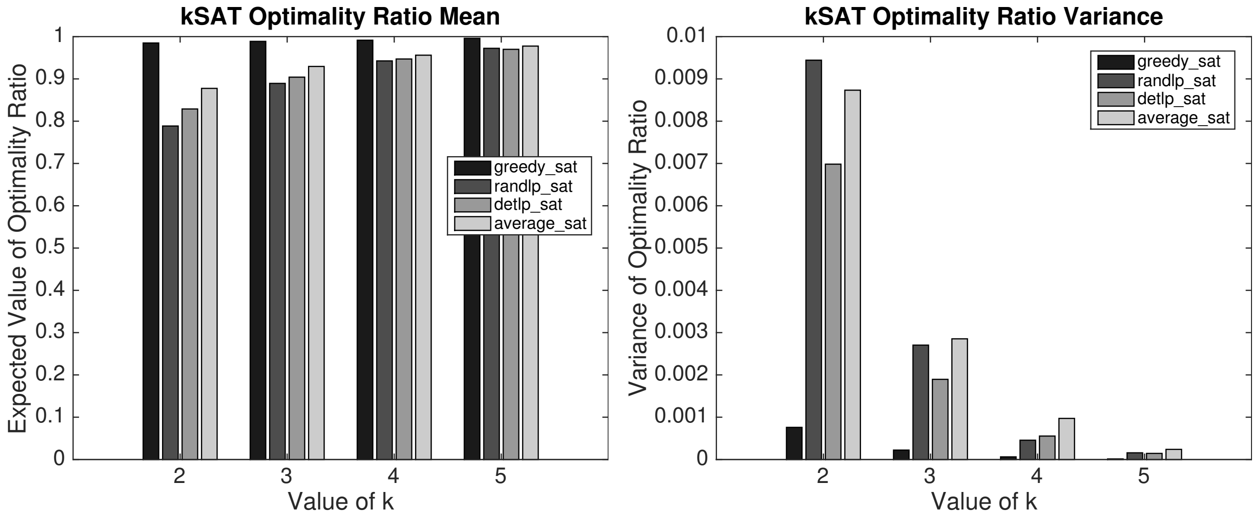


Figure 3.3: Computed over ratio of performance to optimal performance across random inputs with consistent parameters.

In figure 3.3, we can see that *greedy_sat* outperforms the other algorithms pretty handily, but as the value of *k* increases we their performance begins to converge. Additionally, for the variance, we might expect the stochastic algorithms to have much higher variance, because the stochasticity of the inputs is compounded by that of the algorithm. In fact, that doesn't appear to be the case, with *randlp_sat*, *detlp_sat*, *average_sat* all having quite similar values for variance. This implies that our random algorithms may actually be sufficient in practice. Additionally, we see that the *average_sat* just does worse that the *greedy_sat*. Speculatively, if the other three algorithms were competitive with *greedy_sat*, we may have seen an improvement.

3.2 Future Work

Ultimately, we didn't have the resources to really investigate a huge variety of parameters. In the future, if we were to work on this algorithm, there are quite a few additions we could make. Generating more data would help us identify if there are any obviously discernible features and allow us to have a greater chance of generating interesting inputs that the neural network might actually assign to different algorithms. Additionally, there are many more interesting MAX SAT algorithms that may have performance that is capable of matching up with the greedy algorithm. Having other competitive algorithms would likely actually strengthen our ensemble algorithm. One final augmentation to our neural network is to implement more features, potentially from the Alfonso and Manthey's paper on random decision forests for SAT solving [1].

References

[1] Enrique Alfonso and Norbert Manthey. New cnf features and formula classification. In Daniel Le Berre, editor, *POS-14. Fifth Pragmatics of SAT workshop*, volume 27 of *EPiC Series in Computing*, pages 57–71. EasyChair, 2014.

[2] J. Chen, D. K. Friesen, and H. Zheng. Tight bound on johnson's algorithm for maximum satisfiability. *J. Comput. Sys. Sci.*, 58:622–640, 1999.

[3] D. S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. System Sci.*, 9:256–278, 1973.

[4] M. Poloczek and G. Schnitger. Randomized variants of johnson's algorithm for max sat. *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 656–663, 2011.

[5] Anke van Zuylen. Simpler 3/4-approximation algorithms for max sat. *Lecture Notes in Computer Science*, 7164:188–197, 2012.