Experimentation with DPLL and WalkSAT

CS271 Final Project

December 9, 2018

Abhishek Mangla
amangla@uci.edu

Young-Taek Oh
youngto1@uci.edu

Daniel Yao
dsyao@uci.edu

Introduction

In computer science, boolean satisfiability (SAT) is one of the most well-known NP-Complete decision problems; this means there is no known polynomial time algorithm that can solve this type of problem. The problem is given in a propositional logic formula such as:

$$(A \vee \underbrace{B}_{\text{literal}} \vee C) \wedge (\underbrace{D \vee \neg C \vee \neg A}_{\text{clause}}) \wedge (\underbrace{\neg E}_{\text{unit clause}})$$

in order decide whether on not there exists assignments to variables A through E that make the whole expression true. The expression written above is in a special form called Conjunctive Normal Form (CNF) which means each literal within a clause is in a disjunction and each clause in the expression is in a conjunction. Any negation can only be on a single literal. We can safely assume inputs are always in CNF form because it can be shown that any logic formula can be written in CNF. Many challenges in artificial intelligence such as locomotion, decision-making for robots, and planning agents can be abstracted and simplified to a propositional logic form. Therefore, the ability to solve this problem has a great impact on several applications of computer science and real-world problems. There are many types of variations of satisfiability (SAT) problems, but we focus on 3-SAT in which clauses can have no more than 3 literals. It can be shown by algorithm reductions that 3-SAT reduces to general SAT so we can simplify the problem space and run our experiments on 3-SAT.

Even though 3-SAT is most well known as a decision problem, we can easily transform it into a search problem by reframing the problem to *find* a solution to a given propositional logic formula instead of just verifying if a solution exists. In this paper, we attempt to solve the 3-SAT search problem instead of the decision problem.

Previous Work and Methods

Research into SAT solvers has usually taken one of two routes: backtracking search or stochastic local search. The most well known backtracking search method was developed in the 1960s known as the DPLL (Davis-Putnam-Logemann-Loveland) algorithm [1]. DPLL is a complete algorithm which indicates it will find a solution if it exists and will exhaustively search the entire search space. Some optimizations are included in modern implementations of DPLL such as unit propagation, pure literal elimination, and taking advantage of heuristics to decide which branch of literals to search first. Some advanced techniques that have garnered attention because of competitive performance in SAT contests are modified DPLL algorithms that attempt to find the root conflicts upon assignments to efficiently resolve and restart the search. Called Conflict-Driven Clause Learning (CDCL), its main advantage is derived from non-chronological backtracking which allows the algorithm to skip a multitude of pointless searching while still maintaining the important property of completeness [3].

The other type of SAT solvers implement stochastic local search. This category of solvers are not complete, so they are not guaranteed to find a solution even if one exists, but they typically return solutions much faster than backtracking search algorithms. Greedy SAT (GSAT) starts by assigning a random value to each literal in the formula. If the assignment satisfies all clauses, the algorithm returns the assignment and ends. If the assignment fails, a literal's boolean value is flipped and the new assignment is checked again. GSAT selects which variable to flip based on which one will minimize the number of unsatisfied clauses in the new assignment. WalkSAT is another local search algorithm different from GSAT in that it first randomly picks a clause which is unsatisfied by the current assignment, then flips a variable within that clause. The variable is picked, with some probability, that will result in the fewest previously satisfied clauses becoming unsatisfied. GSAT and WalkSAT have versions that reinitialize with a new random assignment if no solution has been found after a timeout period due to the algorithm's natural tendency to become stuck in a local minima of numbers of unsatisfied clauses.

For our project, we first examine the backtracking search algorithm DPLL and its performance characteristics on varying benchmark parameters. We perform similar experiments for WalkSAT with the additional statistic of accuracy because WalkSAT is not guaranteed to find any solutions. Then we develop and test our own version of WalkSAT with various sets of optimal probability values and interpolate a model to find the optimal number of max_flips that guarantees a solution with a high probability. If WalkSAT fails, we accept an incomplete result because DPLL is too time consuming to verify solutions. We also tested different ways of selecting symbols to conduct branch searches in DPLL, but the normal DPLL algorithm still held a better overall performance.

Methodology and Results

DPLL

This complete, exponential algorithm performs a depth-first search of the assignment tree with the addition of some basic heuristics to guide the order of tree traversal. The first heuristic the algorithm examines is the pure symbol heuristic. A symbol is pure if it occurs as the same literal (negated or non-negated) in all the clauses of the formula, meaning the symbol can have only 1 assignment without conflicts in other clauses. Assigning pure symbols simplifies the propositional logic problem and is better than randomly picking symbols. Once the algorithm exhausts all pure symbol assignments, it employs the unit clause heuristic which inspects if any clause has only one literal that has not been assigned any value. After propagating unit clause symbol assignments throughout the formula, the only room for optimization or change is the method of picking symbols after resolving pure symbols and unit clauses. The regular DPLL algorithm picks random symbols to explore and we assess the performance of this algorithm in the following experiments.

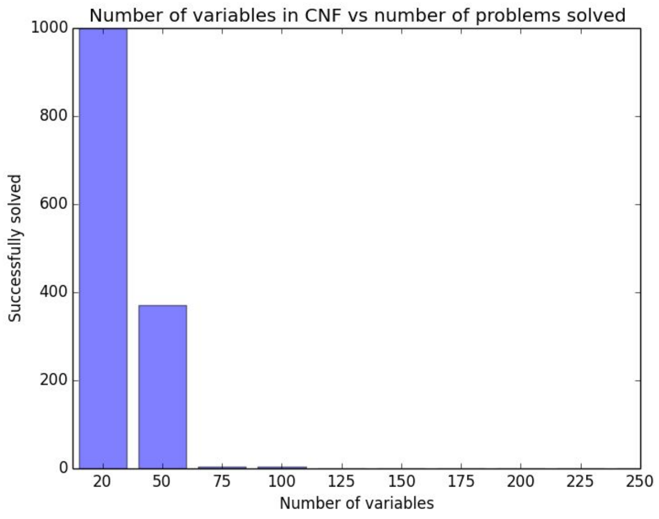

Fig. 1 Experiment 1

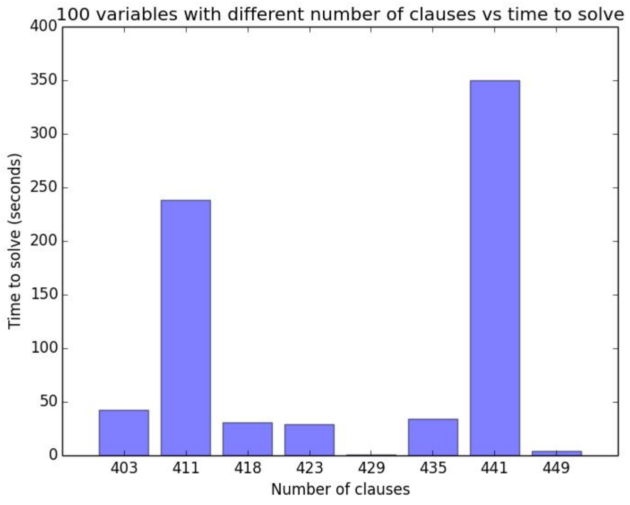

Fig. 2 Experiment 2

Experiment 1 - DPLL success with timeouts

We used the Uniform Random-3-SAT benchmarks labeled "uf-..."[1]. Each benchmark has a static number of variables and clauses and 1000 test cases for each configuration. A timeout of 1 second was set in our DPLL algorithm and then we counted the number of problems out of the provided 1000 test cases that satisfiable solutions were found. Figure 1 displays problems that contain 20 variables and 91 clauses can all be solved in less than 1 second. The benchmark for 50 variables and 218 clauses was able to find solutions 40% of the time given the short 1-second timeout. More complex benchmarks that have greater than 50 variables were essentially unsolvable in the given timeout. This experiment clarifies the exponential runtime of the DPLL algorithm.

Experiment 2 - DPLL performance with varying number of clauses

We used the Random-3-SAT Instances and a fixed variable count of 100. We attempted to find a correlation between number of clauses and the runtime of the DPLL algorithm using a range of 403 clauses to 449 clauses. The runtime for solving 100 variable CNF problems is on average more than a minute and so 5 test cases were used for each benchmark. The resulting data found an ambiguous correlation between number of clauses and runtime. 441 clause test cases had a higher runtime than 411 clauses, but it is not clear if this is an anomaly or a monotonic trend. We need better test cases, perhaps for a lower variable count in order to perform this experiment faster.

Experiment 3 - DPLL performance with most frequent symbol picked next

In the next experiment, we modified the original DPLL algorithm's symbol selection method from simply picking the next unassigned symbol to selecting the symbol that occurs most frequently and currently lacks an assignment. We call this a modified DPLL and a time performance comparison for finding solutions between the modified DPLL and the original DPLL is shown below.

[1]SATLIB Solvers Collection:
https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html?fbclid=IwAR0V82AEcRWMX-HcQH-4U0dI3sbyMbMqu3cAe2Vs0KTSLkNiyRWXAObxpPo

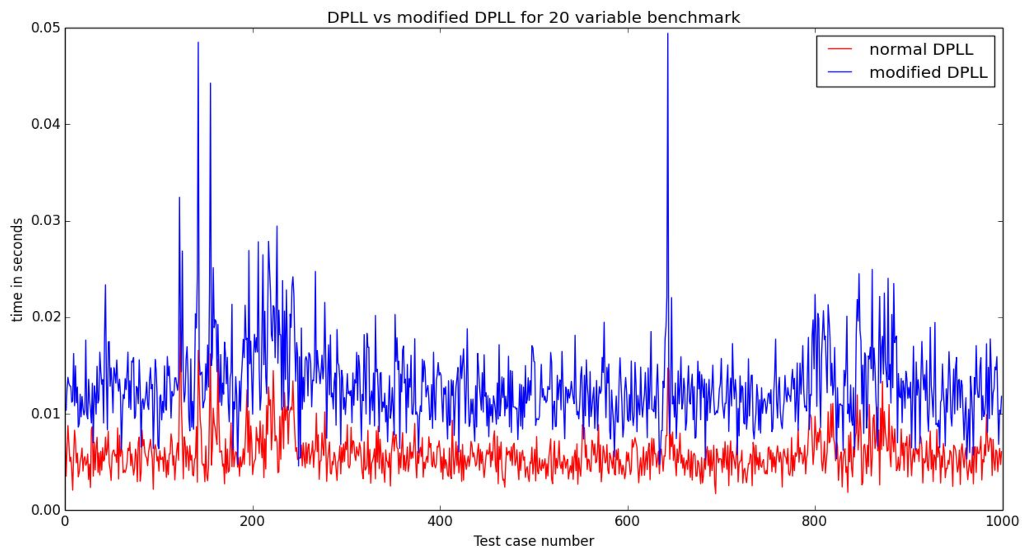

Fig 3. Our modified DPLL algorithm's most frequent symbol selection has a worse time performance than the random symbol selection of normal DPLL algorithm in 20 variable benchmark

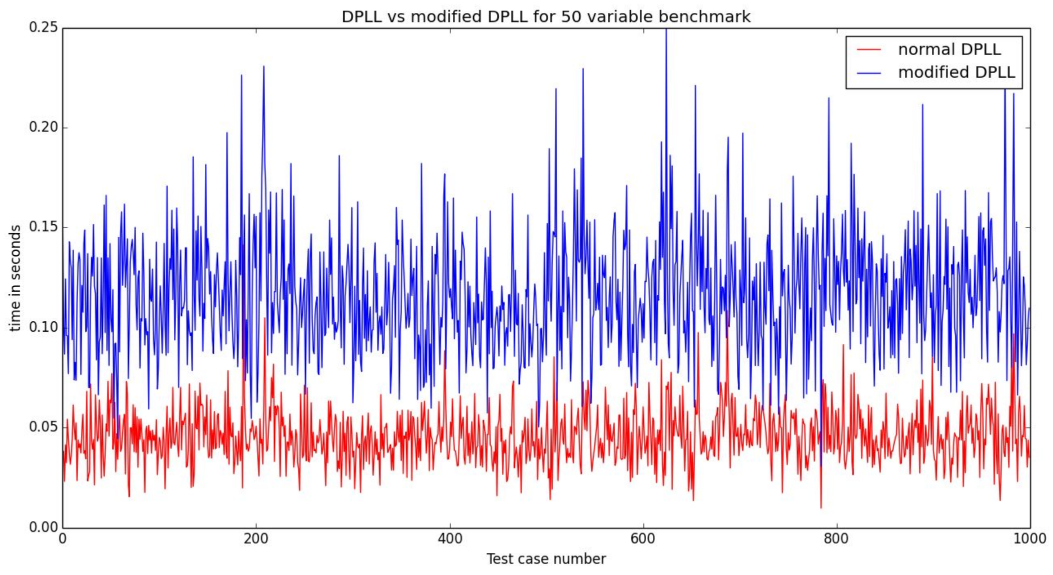

Fig 4. Our modified DPLL algorithm's most frequent symbol selection has worse time performance than the random symbol selection of normal DPLL algorithm even in 50 variable benchmark

Our DPLL experiments indicate that simple heuristics for symbol selection are not enough to achieve drastic improvements. We hypothesized that the most frequent symbol selection would perform better than random symbol selection because we could quickly resolve the most number of symbols in a problem greedily, but it turns out that for 20 variable and even 50 variable test cases, this methodology does not improve results but instead worsens time performance. We also observed an ambiguous correlation between number of clauses and time complexity using DPLL.

Experiment 4 - DPLL with random symbol selection

We also tried DPLL with random symbol selection and found that its mean time performance was worse than most frequent symbol selection and next symbol selection. However on some test cases, random symbol selection DPLL was visibly better than normal DPLL which suggests that perhaps a hill-climbing approach to DPLL could yield better results. With some probability, we could explore a random symbol or otherwise explore the next unassigned symbol.

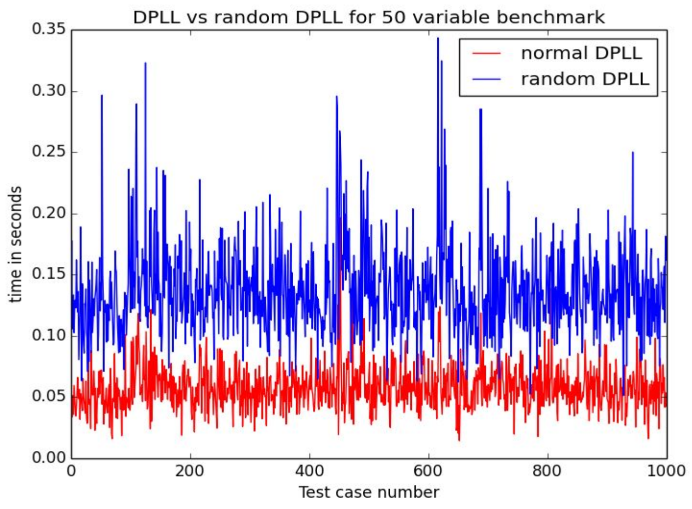

Figure 4A

WalkSAT

WalkSAT is an incomplete local search algorithm which may not always yield a solution. However despite its shortcomings, when a solution exists within the constraints, WalkSAT is significantly quicker than DPLL. WalkSAT operates by first generating a random truth assignment as a temporary model which includes unsatisfied clauses. From the temporary model, a random unsatisfied clause is selected and with a probability, a variable is chosen to ensure the fewest previously satisfied clauses from becoming unsatisfied. When picking a variable at random, WalkSAT assures that there is some chance an incorrect assignment is corrected. As a method of avoiding the local minima, the algorithm may restart with a new random truth assignment. Therefore WalkSAT can discover a solution quickly, but is not guaranteed to find a solution and cannot definitively prove that no solution exists.

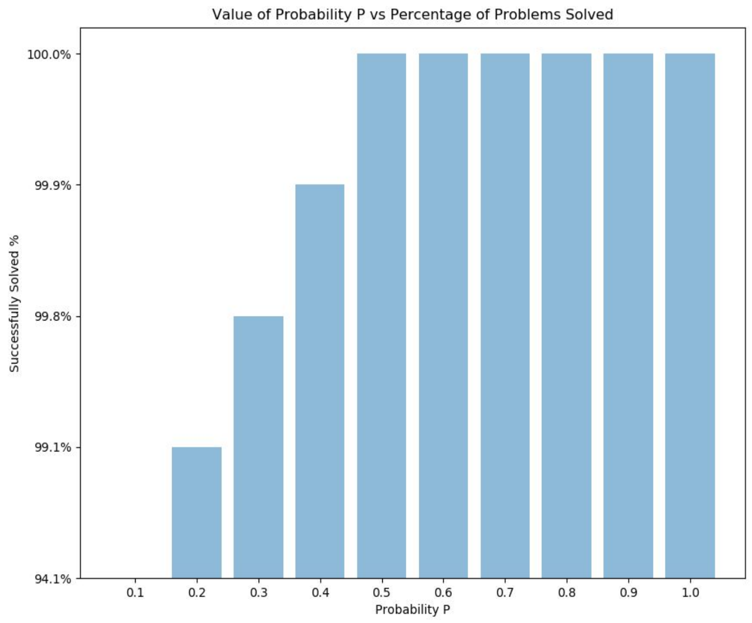

Fig. 5

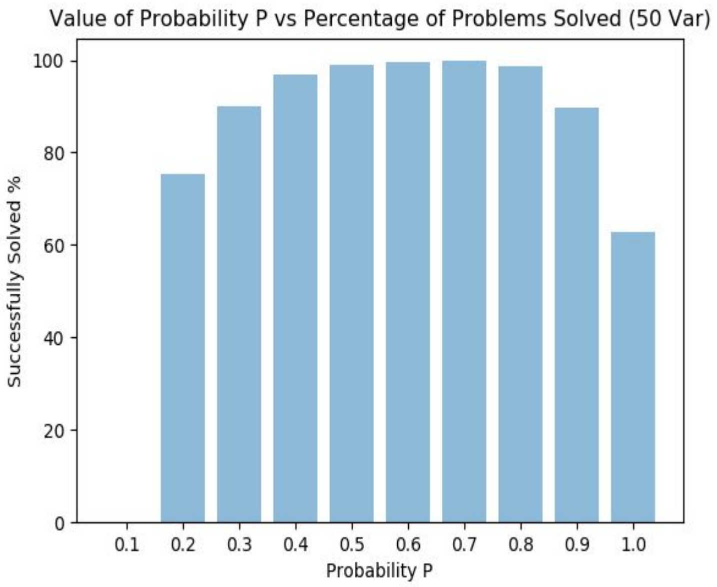

Fig. 6

Experiment 1 - Finding the best probability for WalkSAT

To stay consistent, we used the Uniform Random-3-SAT benchmarks as explained in the DPLL Experiment 1 section. No timeout was set in the WalkSAT algorithm. Out of the 1000 test cases, we calculated the success rate of finding satisfiable solutions for each probability value. From Fig. 5 the probability value 0.2 had a 99.1% success rate in finding solutions with 20 variables and 91 clauses problems. Furthermore, the WalkSAT algorithm had a 100% success rate starting from p = 0.5 to 1.0 for problems with 20 variables and 91 clauses. However, when we ran 1000 test cases for each probability value for problems with 50 variables and 218 clauses, the optimal probability value that led to the highest success rate in finding solutions was 0.7 as shown in Fig 6. We acknowledge that the WalkSAT algorithm is an incomplete algorithm as the highest probability does not necessarily guarantee 100% success rate as shown in Fig. 6, but from this experiment we noticed that the success rate in finding solutions is relatively higher as the probability increases.

Experiment 2 - Creating a model for number of max flips based on problem size

For Experiment 2, our goal was to find the optimal max flips for the WalkSAT algorithm as we found the optimal probability value as 0.7 from Experiment 1. There are 10 different uniform random-3-SAT benchmarks ranging from 20 variables to 250 variables with a fixed number of clauses for each variable benchmark.

We ran 20 test cases for each variable benchmark from max_flips = 1000 to 20,000 incrementing by 1000 each iteration with a constant probability = 0.7 to calculate the success rate of each max_flips value. The full result is shown in Table 1 in the Appendix. From this experiment, we examined a trend of higher number of max_flips required for a higher success rate as variables increased per benchmark. This proves that we needed a model that adaptively calculates the optimal max_flips for different variable counts because even though infinitely large max_flips will eventually find a solution, knowing the smallest number of flips with a relatively high success rate will result in a better time complexity. As shown in Table 1, higher max_flips correlated to the success rate of solving satisfiable problems. However, we realized that after the 100 variables benchmark, a max_flips value greater than 20,000 was needed to achieve better results. Based on this data and through trial-and-error, we created a model:

$$\text{max\_flips} = LN(\#variables^4)(\#variables)(15)$$

We acknowledge that this model computes an estimated # of max_flips based on our data set only and max_flips will become incredibly high when a substantial number of variables are encountered.

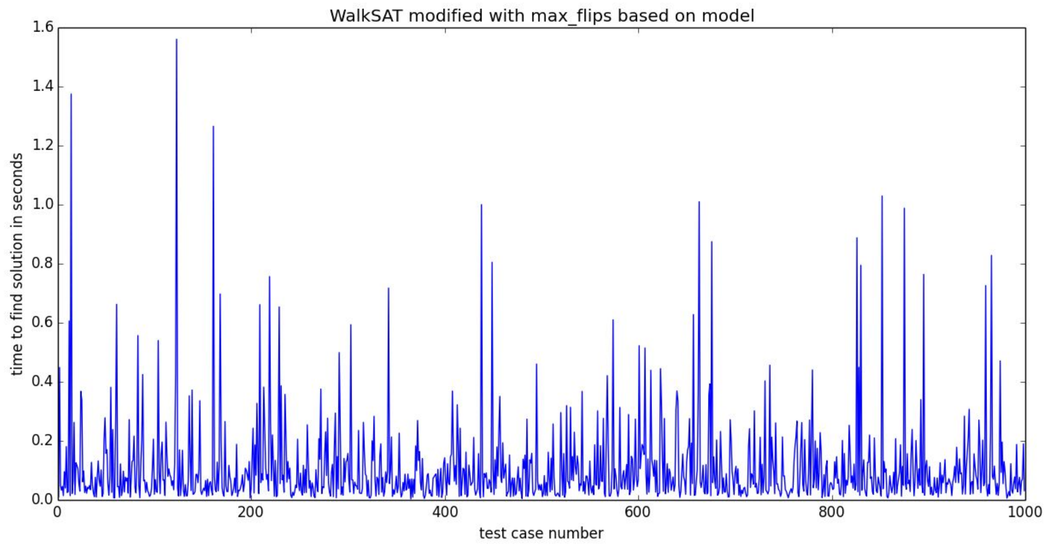

Fig. 7 This graph shows the time it took to solve a 50 variable benchmark using our version of WalkSAT.

Experiment 3 - WalkSAT Modified (use model from WalkSAT Experiment 2)

Based on the results from our experiments with DPLL and WalkSAT, we have 2 important observations that guided us in formulating an optimal algorithm that combines the advantages of both approaches.

1) DPLL is complete but computationally intractable. The random symbol selecting DPLL works better than our proposed DPLL method of using most frequent symbols.
2) WalkSAT is not complete, but given a certain number of max_flips, there is a certain probability that it will succeed in solving a given problem with x variables and y clauses.

It is infeasible for most problem sizes to do backtracking search, so we will compromise the ability to determine unsatisfiable problems and even find satisfiable problems 100% of the time by using WalkSAT with the best parameters possible. We discovered an optimal probability = 0.7 and derived a model that maps the number of variables in a test case to the optimal number of flips so that we have a reasonable chance of finding the solution. To test the performance of our modified WalkSAT algorithm, we contrast the time performance and percentage of solutions they find for the 50 variable benchmark as presented in Fig. 7.

Perhaps one of the most interesting results of this experiment was that out of 1000 satisfiable test cases, WalkSAT solved all 1000, but given the same time that WalkSAT took to solve each test case, DPLL solved only 110 problems and timed out for the other 890 test cases. The table below summarizes this stark contrast in performance and usefulness.

Algorithm	**% of test cases solved**
WalkSAT with 11736 flips	100%
DPLL with timeout based on WalkSAT run duration	11%

Conclusion and Future Work

For this project, we chose WalkSAT Modified as our SAT solver because we were willing to trade off a complete solution for a program that gave results in a reasonable amount of time. We tried to equip our WalkSAT algorithm with just the right parameters so that it would have the best leniency to find a solution if it exists, otherwise return none. Based on our Experiment 3 in the WalkSAT section, it is clear that an algorithm that solves 100% test cases in *x* time is better than a complete algorithm that solves 11% of test cases in the same time.

Other learnings we found were the results from experiment 4 of the DPLL section; we discovered that randomly selecting symbols can sometimes lead to better time performance for finding solutions than normal DPLL, even though it performs worse on average. This suggests that we could try probing random symbols with a small probability than simply just looking at the next symbol.

We also found a need for more test cases to find a correlation between the number of clauses and complexity of the SAT problem instance. As the WalkSAT algorithm relies on probability p and # of max_flips, we experimented on finding optimal values for both probability and max_flips as described in Experiment 1 and 2 under the WalkSAT section. We estimated an optimal probability = 0.7 to be the most efficient value and came up with a model:

$$\text{max\_flips} = LN(\#variable^4)(\#variables)(15)$$

that computes # of max_flips based on # of variables. This model is derived from our observations and multiple trial-and-error attempts. Our estimations on probability and max_flips could be more accurate and precise if we had a larger dataset with a wider range of variables.

Finally, in a time-permitting future, we would have further examined and implemented Conflict-Driven Clause Learning (CDCL) because it's a sound and complete algorithm like DPLL but takes advantage of non-chronological back jumping. Theoretically this means its runtime would outperform the standard DPLL algorithm and it would have been fruitful to compare its performance with DPLL.

References

1. Davis, M.; Logemann, G.; and Loveland, D. 1962. A machine program for theorem-proving. *Commun. ACM* 5(7):394–397.
2. Henry Kautz and B. Selman (1996). Pushing the envelope: planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence* (AAAI'96), pages 1194–1201.
3. Jain, A., Madhyastha H., Prince, C. 2004. An Analysis and Comparison of Satisfiability Solving Techniques. American Association for Artificial Intelligence.
4. J.P. Marques-Silva; Karem A. Sakallah (November 1996). "GRASP-A New Search Algorithm for Satisfiability". Digest of IEEE International Conference on Computer-Aided Design (ICCAD). pp. 220–227. doi:10.1109/ICCAD.1996.569607
5. Russell, S., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition.

Appendix

Table 1: Percentage of success of finding satisfiable solutions given the number of max_flips and variable size of problems for the WalkSAT Algorithm

	20 vars	50 vars	75 vars	100 vars	125 vars	150 vars	175 vars	200 vars	225 vars	250 vars
1000 flips	100	60	75	10	20	0	0	0	5	0
2000 flips	100	95	70	35	20	0	10	0	0	0
3000 flips	100	85	70	35	40	20	0	0	15	15
4000 flips	100	85	85	45	20	20	15	0	5	15
5000 flips	100	90	75	50	40	40	5	5	25	5
6000 flips	100	95	95	55	60	20	20	10	5	10
7000 flips	100	90	80	65	40	40	25	10	5	5
8000 flips	100	100	95	65	40	60	20	15	15	20
9000 flips	100	100	95	65	80	40	30	20	30	15
10000 flips	100	100	100	60	60	40	25	10	15	15
11000 flips	100	100	95	70	80	60	20	20	10	15
12000 flips	100	100	90	70	80	60	35	10	30	10
13000 flips	100	100	90	75	80	60	35	10	30	15
14000 flips	100	100	95	85	80	40	30	15	30	10
15000 flips	100	100	95	75	80	60	30	5	40	35
16000 flips	100	100	100	80	100	60	40	10	35	15
17000 flips	100	100	100	85	100	40	25	15	25	25
18000 flips	100	100	95	75	100	20	40	25	35	30
19000 flips	100	100	100	90	80	80	40	40	55	30
20000 flips	100	100	100	85	60	60	40	35	40	30

Table 2: Class Benchmark Results using our WalkSAT modified algorithm

N = # of variables	# of problems Found Solution	# of Problems No Solution Found	Runtime stats (min, max, mean, median, standard deviation) in seconds
100	0	100	(62.32, 76.43, 67.11, 68.43, 0.124)
200	0	200	(154.76, 62.32, 62.32, 62.32, 0.124)
500	n/a	n/a	n/a
1000	n/a	n/a	n/a
2000	n/a	n/a	n/a
5000	n/a	n/a	n/a
10000	n/a	n/a	n/a
20000	n/a	n/a	n/a
50000	n/a	n/a	n/a
100000	n/a	n/a	n/a