CS 8803 LCS
Project 1 Final

Maya Iyer
Email: mayaiyer@gatech.edu
GTID: 903572496

October 1, 2023

Honor Code: I commit to uphold the ideals of honor and integrity by refusing to betray the trust bestowed upon me as a member of the Georgia Tech community.

1 Source Code

GitHub link

The DPLL algorithm proceeds as follows:

1. If the CNF formula is empty, then the current assignment is a satisfying assignment and the algorithm returns true.
2. If the CNF formula contains a unit clause, then the algorithm assigns the truth value to the literal in the unit clause that makes the clause true. The algorithm then updates the CNF formula by removing all clauses that contain the literal that was assigned a truth value.
3. If the CNF formula contains a contradiction, then the current assignment is not a satisfying assignment and the algorithm returns false.
4. Otherwise, the algorithm selects a variable that is not assigned a truth value. The algorithm then recursively calls itself twice, once with the variable assigned to true and once with the variable assigned to false.

I made three heuristic to determine the next unassigned proposition to proceed with:

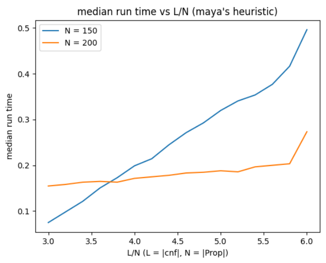
1. A random heuristic - in which the unassigned proposition is decided randomly
2. A two clause heuristic - in which the unassigned proposition with maximum occurrences in 2-clauses is chosen

3. Maya's heuristic - Jerslow-Wang heuristic, which gives an exponentially higher weight to literals in shorter clauses

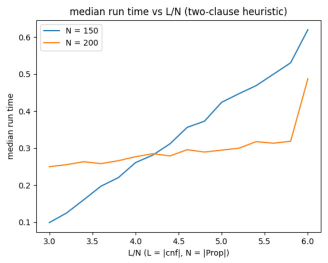
$$\sum_{x \epsilon c, c \epsilon F} 2^{-|c|}$$

2 Compute time and number of DPLL calls on random formulas as a function of L/N

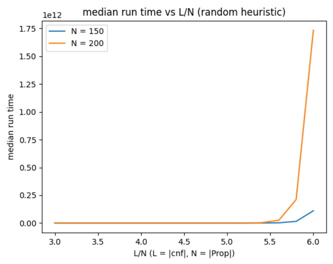
Figure 1 shows compute time on random formulas as a function of L/N, where L is the number of clauses, and N is the number of propositions.



(a) my heuristic



(b) two-clause heuristic



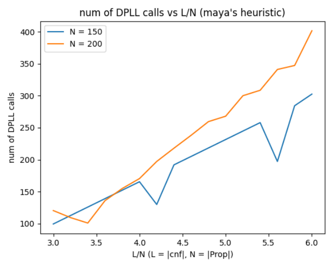
(c) random heuristic

Figure 1: Compute time vs L/N

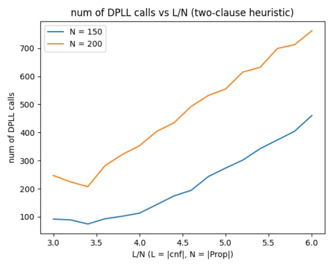
Figure 2 shows number of DPLL calls on random formulas as a function of L/N, where L is the number of clauses, and N is the number of propositions.

Choosing a variable randomly in step 4 of the DPLL algorithm can lead to a decrease in performance for the following reasons:

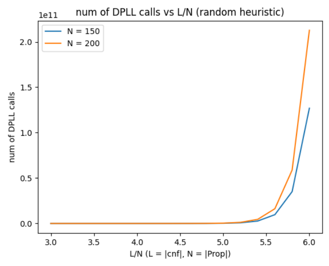
- Randomness can lead to unnecessary exploration of the search space. If the algorithm randomly selects a variable that is not relevant to the satisfiability of the formula, then the algorithm may need to explore a large



(a) my heuristic



(b) two-clause heuristic



(c) random heuristic

Figure 2: number of DPLL calls vs L/N

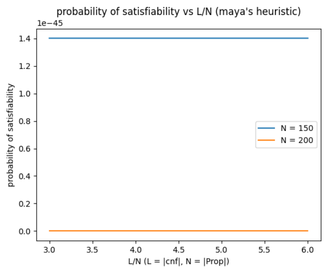
portion of the search space before finding a satisfying assignment or determining that the formula is unsatisfiable.

- Randomness can lead to the algorithm getting stuck in local optima. A local optimum is an assignment that satisfies all of the clauses in the formula, but there may be other assignments that satisfy more clauses. If the algorithm randomly selects a variable that leads to a local optimum, then the algorithm may not be able to find a better assignment.
- Randomness can lead to the algorithm taking a long time to find a solution. The number of possible variable assignments can be very large for large formulas. If the algorithm randomly selects variables, then it may take a long time to find a satisfying assignment or determine that the formula is unsatisfiable.

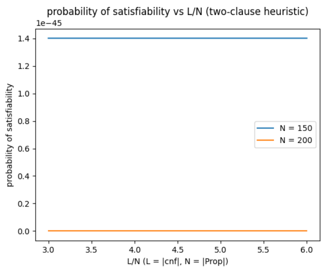
By using a heuristic to select variables in step 4, the DPLL algorithm can be made more efficient and can find solutions to larger formulas more quickly, as shown in the figures.

3 Probability of satisfiability of random formulas as a function of L/N

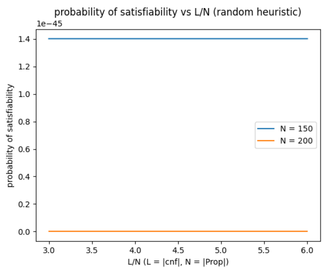
Figure 3 shows the probability of satisfiability on random formulas as a function of L/N, where L is the number of clauses, and N is the number of propositions.



(a) my heuristic



(b) two-clause heuristic



(c) random heuristic

Figure 3: probability of satisfiability vs L/N

The more propositions there are in a formula, the less likely the formula is to be satisfiable. This is because the number of possible variable assignments grows exponentially with the number of propositions. For example, a formula with 10 propositions has 1024 possible variable assignments, while a formula with 20 propositions has 1,048,576 possible variable assignments.

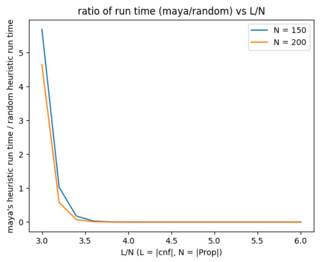
For a random formula with a given number of propositions, the probability that the formula is satisfiable is given by the following formula:

$P(SAT) = 2^{-n}$, where n is the number of propositions in the formula.

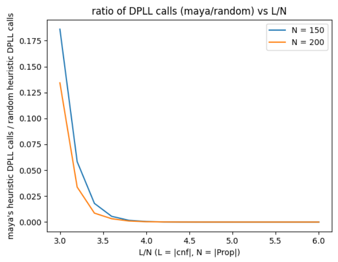
This formula shows that the probability of satisfiability decreases exponentially with the number of propositions. For example, a formula with 10 propositions has a probability of satisfiability of 1/1024, while a formula with 20 propositions has a probability of satisfiability of 1/1,048,576. This means that as the number of propositions in a formula increases, it becomes more and more unlikely that the formula will be satisfiable. It is important to note that the above formula only gives the probability of satisfiability for random formulas. For specific formulas, the probability of satisfiability may be higher or lower than the probability given by the formula.

4 My heuristic versus the random heuristic

Figure 4 illustrates the comparison in performance of my heuristic to the random one.



(a) Compute time



(b) number of DPLL calls

Figure 4: My heuristic versus the random heuristic

A heuristic that gives an exponentially higher weight to literals in shorter clauses can be better than randomly choosing a variable because it is more likely to lead to a satisfying assignment more quickly.

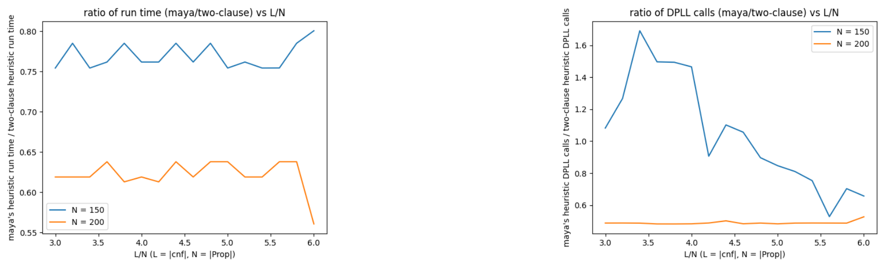
This is because shorter clauses are more likely to be unit clauses, which are clauses that contain only a single literal. Unit clauses can be assigned a truth value immediately, which can reduce the size of the search space and make it more likely that the algorithm will find a satisfying assignment.

Additionally, shorter clauses are more likely to be involved in conflicts, which are clauses that are not satisfied by the current assignment. When a conflict is detected, the algorithm can backtrack and try a different assignment. By focusing on shorter clauses, the algorithm is more likely to detect conflicts early, which can save time and improve the overall performance of the algorithm.

5 My heuristic versus the two-clause heuristic

Figure 5 illustrates the comparison in performance of my heuristic to the two-clause heuristic.

In a 3-SAT formula, most clauses will be of length 2 after a few iterations of DPLL. In order to maximize the amount of information factored into the



(a) Compute time

(b) number of DPLL calls

Figure 5: My heuristic versus the two-clause heuristic

heuristic, we should use a heuristic that is better at identifying all the clauses the literals are present in.

6 Findings

Heuristics can affect the performance of a DPLL implementation in a number of ways. Heuristics can help the algorithm to avoid exploring unnecessary parts of the search space. For example, if the algorithm selects a variable that is not important for the satisfiability of the formula, then the algorithm may need to explore a large portion of the search space before finding a satisfying assignment or determining that the formula is unsatisfiable. By using a heuristic to select variables, the algorithm can focus on the parts of the search space that are most likely to contain a satisfying assignment.

Heuristics can also help the algorithm to find solutions more quickly. The number of possible variable assignments can be very large for large formulas. If the algorithm randomly selects variables, then it may take a long time to find a satisfying assignment or determine that the formula is unsatisfiable. By using a heuristic to select variables, the algorithm can be more likely to find a satisfying assignment more quickly.