CS301

2022-2023 Spring

Project Report

Group 129

Nural Kaan Özdemir

1. Problem Description

The Maxiumum-2-Satisfiability problem is a widely known optimization problem in computer science. Problem can be described intuitively as follows: Given A set of m clauses $C_1, C_2, ..., C_m$ over n Boolean valued variables X_n, where each clause depends on two distinct variables from X_n; a positive integer k with $k <= m$. Is there an assignment of Boolean values to the variables X_n such that at least k distinct clauses take the value true under the assignment? Maxiumum-2-Satisfiability problem is a variation of the Boolean satisfiability problem (SAT) that deals with Boolean expressions that are in conjunctive normal form (CNF) and contains at most two literals per clause. Goal of this problem is to find an assignment to the variables of the formula that will satisfy the maximum number of clauses simultaneously, rather than simply determining whether a solution exists or not.

This theorem was proven NP-Complete. According to Williams[1], since 3-SAT problem reduces to Max-2-SAT, it follows that Max-2-SAT (as a decision problem) is an NP-complete problem.

Several examples can be given about the applications of the Maxiumum-2-Satisfiability problem. Circuit design is one of the application fields of this problem. Artificial Intelligence and Graph Theory are also among the application fields of this problem.

2. Algorithm Description

a. Brute Force Algorithm

This brute force algorithm described below is my own implementation. First, our algorithm declares a variable max_satisfied to store the maximum number of clauses satisfied. Initially, this value is initialized to 0. Then, algorithm starts iterating through all possible truth value assignments of the given variables. In each iteration, it counts number of clauses satisfied via the current truth value assignments. If number of clauses satisfied in the current iteration is greater than max_satisfied, algorithm updates max_satisfied to the number of clauses satisfied in the current iteration. After this iteration finishes, algorithm starts iterating through all possible truth value assignments of the given variables again. In each iteration, it counts number of clauses satisfied via the current truth value assignments. If number of clauses satisfied in the

[1] Williams, Ryan. “Maximum 2-Satisfiability (2004; Williams) - Massachusetts Institute of ...” Accessed April 28, 2023. https://people.csail.mit.edu/rrw/williams-max2sat-encyc.pdf.

current iteration is equal to the max_satisfied value, it prints out variables with their corresponding truth values. After second iteration ends, algorithm returns maximum number of clauses satisfied in the formula. Pseudocode of the brute force algorithm is provided below:

```
Initialize variable max_satisfied = 0
Start iterating over all possible truth assignments:
    In each iteration:
        Assign corresponding Boolean values to variables
        Count number of clauses satisfied with given values to variables
        If number of clauses satisfied currently > max_satisfied
            Update max_satisfied to number of clauses satisfied currently
Start iterating over all possible truth assignments:
    In each iteration:
        Assign corresponding Boolean values to variables
        Count number of clauses satisfied with given values to variables
        If number of clauses satisfied currently == max_satisfied
            Print the variables and their corresponding values
Return max_satisfied
```

b. Heuristic Algorithm

For heuristic algorithm part, I've implemented a randomized approach algorithm myself. This algorithm iterates through clauses a specified amount of time (*num_iterations*) and in each iteration, it generates a random assignment by selecting either 1 or 0 for each variable. Then, it calculates the number of clauses satisfied and compares it with the highest number of clauses satisfied up to this point. If it is greater than highest number of clauses satisfied, it replaces highest number of clauses satisfied with current result and stores the truth assignments of the variables that achieves this result. In the end, it returns the number of maximum clauses satisfied and corresponding truth assignments. Below is the pseudocode for this algorithm:

```
Initialize best_assignments
Initialize best_satisfaction
Start iteration up to num_iterations
```

assignment = Assign 0's and 1's to variables randomly

satisfaction = Calculate how many clauses are satisfied with current assignment

if *satisfaction* > *best_satisfaction*:

best_assignments = *assignment*

best_satisfaction = *satisfaction*

else if *satisfaction* == *best_satisfaction*:

add *assignment* to *best_assignments*

return *best_assignments* and *best_satisfaction*

3. Algorithm Analysis

a. Brute Force Algorithm

To prove our brute force algorithm works correctly, in a form of a theorem, let's assume that our brute force algorithm does not always find the maximum number of satisfied clauses for a given boolean formula in CNF form. Suppose there exists a boolean formula in CNF form with n variables, where our algorithm outputs a solution that satisfies k clauses, but there is another solution that satisfies more clauses. Let's call the better solution found by some other method "Solution A", which satisfies m clauses, where $m > k$. Because Solution A satisfies more clauses than our algorithm's solution, we can conclude that Solution A is the maximum number of satisfied clauses that can be achieved for this formula. This, however, contradicts our premise that our algorithm always finds the largest number of satisfied clauses. As a result, our assumption is incorrect, and our method must always discover the maximum number of satisfied clauses for any boolean formula in CNF form. Thus, we can say that the Max-2-SAT problem can be solved correctly using our brute force algorithm.

To find out the worst-case time complexity of our algorithm, let's assume that there are n variables and m clauses. Brute force algorithm would require checking 2^n possible assignments of variables and their negations, and for each assignment, checking m clauses. This leads to a time complexity of $O(2^n * m)$, which is exponential.

b. Heuristic Algorithm

To show the correctness of the algorithm, we need to establish two key properties: optimality and soundness. The algorithm aims to find the best satisfying assignment for the given max-2-SAT problem. It does this by iteratively generating random assignments and updating the best assignment and satisfaction score whenever a higher

satisfaction score is achieved. Hence, we can consider this heuristic as an optimal algorithm. The algorithm follows a randomized approach, generating random assignments and evaluating their satisfaction scores. This randomness introduces exploration and diversity into the search process, which helps avoid getting stuck in local optima. Therefore, we can state that this heuristic satisfies soundness property. Based on these properties, we can conclude that the heuristic algorithm correctly solves the problem, finding the best satisfying assignments with a high probability.

The randomized approach algorithm runs for *num_iterations* iterations. Assume that *m* is the number of clauses, *n* is the number of variables and *num_iterations* is the number of iterations. Within each iteration, a random assignment is generated in O(*n*) time, and the satisfaction is calculated in O(*m*n*) time. Therefore, the worst-case time complexity of the algorithm will be O(num*_iterations*(n + m * n)*).

The space consuming factor in this algorithm is storing the best assignments. Consider again *m* is the number of clauses, *n* is the number of variables and *num_iterations* is the number of iterations. In the worst case, if all iterations produce best satisfaction, the space complexity would be O(*num_iterations*n*) as there are *n* variables. Also, we need to consider the storage of the clauses, which will result in O(*m*) as there are *m* clauses. So considering these two factors, overall space complexity of this algorithm will be O(*m* + num*_iterations*n*).

4. Sample Generation (Random Instance Generator)

This sample generating algorithm described below is my own implementation. Random sample generating algorithm requires two vectors to store clauses and variables. They must be passed as reference to the function. It also needs the values for number of variables and number of clauses to be generated, passed by values to the function. First, algorithm generates specified number of variables. Then, it starts generating given number of clauses. Algorithm generates a clause by randomly selecting two variables initially. Then, it randomly negates those variables, or leave them as it is. Finally, it pushes generated clause into the vector provided to store all generated clauses. This algorithm generates one formula at a time containing numerous variables and clauses. However, the function can be called multiple times using a loop to populate many samples. Pseudocode for the sample generation algorithm is as follows:

Loop 1 to number of variables to be generated (inclusive)

Create new variable

Push new variable to passed vector to store variables

Loop 0 to number of clauses to be generated

Randomly select two variables among created variables

Randomly negate selected two variables

Push selected variables to provided vector which stores clauses

5. Algorithm Implementations

a. Brute Force Algorithm

Tested 15 randomly generated samples using the sample generator above. In each iteration, number of variables to be used in the formula are generated randomly in the range 2-7. Also, the number of clauses is generated randomly as well. The range which determines this value is computed based on the number of created variables. If number of created variables is an even number, the range will be number of variables / 2 to number of variables / 2 + 5. Otherwise, the range will be number of variables / 2 + 1 to (number of variables / 2 + 1) + 9. In each iteration, values for number of variables to be created and number of clauses to be created are generated randomly between the specified ranges of them, and they will be passed as values to the sample generator. Finally, the created clause will be evaluated. Result of an iteration with 15 samples is provided below:

----- Iteration 1 -----
Number of variables: 3
Number of clauses: 4
Variables: x1 x2 x3
Clauses: {x1, x2} {!x3, !x1} {!x3, x2} {!x1, x2}
Truth assignment(s) satisfying maximum number of clauses:
x1=0 x2=1 x3=0
x1=1 x2=1 x3=0
x1=0 x2=1 x3=1
Maximum number of clauses satisfied: 4
----- Iteration 2 -----
Number of variables: 5

Number of clauses: 10

Variables: x1 x2 x3 x4 x5

Clauses: {!x2, !x3} {!x1, x4} {!x1, !x5} {x3, x1} {x3, !x5} {x2, x5} {!x5, !x3} {x3, !x2} {x2, !x5} {x1, !x4}

Truth assignment(s) satisfying maximum number of clauses:

x1=0 x2=0 x3=1 x4=0 x5=0

x1=0 x2=1 x3=1 x4=0 x5=0

x1=1 x2=0 x3=0 x4=1 x5=0

x1=1 x2=1 x3=0 x4=1 x5=0

x1=1 x2=0 x3=1 x4=1 x5=0

x1=1 x2=1 x3=1 x4=1 x5=0

Maximum number of clauses satisfied: 9

----- Iteration 3 -----

Number of variables: 5

Number of clauses: 10

Variables: x1 x2 x3 x4 x5

Clauses: {!x4, x3} {!x2, x5} {!x4, x5} {x3, x4} {x4, !x1} {!x3, !x1} {!x5, !x1} {!x3, !x5} {x3, !x2} {!x5, !x1}

Truth assignment(s) satisfying maximum number of clauses:

x1=0 x2=0 x3=1 x4=0 x5=0

Maximum number of clauses satisfied: 10

----- Iteration 4 -----

Number of variables: 3

Number of clauses: 2

Variables: x1 x2 x3

Clauses: {!x1, !x2} {x1, !x2}

Truth assignment(s) satisfying maximum number of clauses:

x1=0 x2=0 x3=0

x1=1 x2=0 x3=0

x1=0 x2=0 x3=1

x1=1 x2=0 x3=1

Maximum number of clauses satisfied: 2

----- Iteration 5 -----

Number of variables: 4

Number of clauses: 9

Variables: x1 x2 x3 x4

Clauses: {x4, x2} {x1, !x3} {x3, x2} {x3, x2} {x3, !x4} {!x1, x4} {x1, !x4} {x3, !x2} {!x4, x2}

Truth assignment(s) satisfying maximum number of clauses:

x1=1 x2=1 x3=1 x4=1

Maximum number of clauses satisfied: 9

----- Iteration 6 -----

Number of variables: 3

Number of clauses: 2

Variables: x1 x2 x3

Clauses: {x2, x1} {!x1, x2}

Truth assignment(s) satisfying maximum number of clauses:

x1=0 x2=1 x3=0

x1=1 x2=1 x3=0

x1=0 x2=1 x3=1

x1=1 x2=1 x3=1

Maximum number of clauses satisfied: 2

----- Iteration 7 -----

Number of variables: 5

Number of clauses: 6

Variables: x1 x2 x3 x4 x5

Clauses: {!x2, x3} {!x5, x4} {!x5, !x3} {x1, !x4} {!x5, !x2} {!x2, x1}

Truth assignment(s) satisfying maximum number of clauses:

x1=0 x2=0 x3=0 x4=0 x5=0

x1=1 x2=0 x3=0 x4=0 x5=0

x1=0 x2=0 x3=1 x4=0 x5=0

x1=1 x2=0 x3=1 x4=0 x5=0

x1=1 x2=1 x3=1 x4=0 x5=0

x1=1 x2=0 x3=0 x4=1 x5=0

x1=1 x2=0 x3=1 x4=1 x5=0

x1=1 x2=1 x3=1 x4=1 x5=0

x1=1 x2=0 x3=0 x4=1 x5=1

Maximum number of clauses satisfied: 6

----- Iteration 8 -----

Number of variables: 5

Number of clauses: 5

Variables: x1 x2 x3 x4 x5

Clauses: {x4, x5} {!x1, !x4} {x4, x5} {!x1, x5} {!x1, !x3}

Truth assignment(s) satisfying maximum number of clauses:

x1=0 x2=0 x3=0 x4=1 x5=0

x1=0 x2=1 x3=0 x4=1 x5=0

x1=0 x2=0 x3=1 x4=1 x5=0

x1=0 x2=1 x3=1 x4=1 x5=0

x1=0 x2=0 x3=0 x4=0 x5=1

x1=1 x2=0 x3=0 x4=0 x5=1

x1=0 x2=1 x3=0 x4=0 x5=1

x1=1 x2=1 x3=0 x4=0 x5=1

x1=0 x2=0 x3=1 x4=0 x5=1

x1=0 x2=1 x3=1 x4=0 x5=1

x1=0 x2=0 x3=0 x4=1 x5=1

x1=0 x2=1 x3=0 x4=1 x5=1

x1=0 x2=0 x3=1 x4=1 x5=1

x1=0 x2=1 x3=1 x4=1 x5=1

Maximum number of clauses satisfied: 5

----- Iteration 9 -----

Number of variables: 3

Number of clauses: 7

Variables: x1 x2 x3

Clauses: {!x2, !x3} {x3, !x2} {!x3, x2} {x2, !x1} {x2, x3} {!x2, x3} {!x3, !x2}

Truth assignment(s) satisfying maximum number of clauses:

x1=0 x2=0 x3=0

x1=0 x2=0 x3=1

Maximum number of clauses satisfied: 6

----- Iteration 10 -----

Number of variables: 5

Number of clauses: 4

Variables: x1 x2 x3 x4 x5

Clauses: {x3, x1} {x5, !x1} {!x1, x4} {x2, x3}

Truth assignment(s) satisfying maximum number of clauses:

x1=0 x2=0 x3=1 x4=0 x5=0

x1=0 x2=1 x3=1 x4=0 x5=0

x1=0 x2=0 x3=1 x4=1 x5=0

x1=0 x2=1 x3=1 x4=1 x5=0

x1=0 x2=0 x3=1 x4=0 x5=1

x1=0 x2=1 x3=1 x4=0 x5=1

x1=1 x2=1 x3=0 x4=1 x5=1

x1=0 x2=0 x3=1 x4=1 x5=1

x1=1 x2=0 x3=1 x4=1 x5=1

x1=0 x2=1 x3=1 x4=1 x5=1

x1=1 x2=1 x3=1 x4=1 x5=1

Maximum number of clauses satisfied: 4

----- Iteration 11 -----

Number of variables: 2

Number of clauses: 3

Variables: x1 x2

Clauses: {!x2, x1} {!x1, !x2} {x1, x2}

Truth assignment(s) satisfying maximum number of clauses:

x1=1 x2=0

Maximum number of clauses satisfied: 3

----- Iteration 12 -----

Number of variables: 5

Number of clauses: 5

Variables: x1 x2 x3 x4 x5

Clauses: {x1, x3} {!x1, x5} {!x3, !x1} {!x4, !x2} {!x4, x5}

Truth assignment(s) satisfying maximum number of clauses:

x1=0 x2=0 x3=1 x4=0 x5=0

x1=0 x2=1 x3=1 x4=0 x5=0

x1=1 x2=0 x3=0 x4=0 x5=1

x1=1 x2=1 x3=0 x4=0 x5=1

x1=0 x2=0 x3=1 x4=0 x5=1

x1=0 x2=1 x3=1 x4=0 x5=1

x1=1 x2=0 x3=0 x4=1 x5=1

x1=0 x2=0 x3=1 x4=1 x5=1

Maximum number of clauses satisfied: 5

----- Iteration 13 -----

Number of variables: 2

Number of clauses: 3

Variables: x1 x2

Clauses: {x1, !x2} {x1, x2} {x1, !x2}

Truth assignment(s) satisfying maximum number of clauses:

x1=1 x2=0

x1=1 x2=1

Maximum number of clauses satisfied: 3

----- Iteration 14 -----

Number of variables: 5

Number of clauses: 4

Variables: x1 x2 x3 x4 x5

Clauses: {x3, !x4} {x4, !x1} {x4, !x3} {x3, !x2}

Truth assignment(s) satisfying maximum number of clauses:

x1=0 x2=0 x3=0 x4=0 x5=0

x1=0 x2=0 x3=1 x4=1 x5=0

x1=1 x2=0 x3=1 x4=1 x5=0

x1=0 x2=1 x3=1 x4=1 x5=0

x1=1 x2=1 x3=1 x4=1 x5=0

x1=0 x2=0 x3=0 x4=0 x5=1

x1=0 x2=0 x3=1 x4=1 x5=1

x1=1 x2=0 x3=1 x4=1 x5=1

x1=0 x2=1 x3=1 x4=1 x5=1

x1=1 x2=1 x3=1 x4=1 x5=1

Maximum number of clauses satisfied: 4

----- Iteration 15 -----

Number of variables: 3

Number of clauses: 5

Variables: x1 x2 x3

Clauses: {!x3, x1} {x1, x3} {x1, !x2} {!x1, x3} {!x3, x2}

Truth assignment(s) satisfying maximum number of clauses:

x1=1 x2=1 x3=1

Maximum number of clauses satisfied: 5

b. Heuristic Algorithm

Using sample generator, created 21 formulas randomly and executed them using our heuristic algorithm to perform an initial testing. Each formula can contain variables in range 2 and 10. Number of clauses each formula has depends on the number of variables it contains. Consider x as the number of variables which a formula has, the number of clauses to be generated for the formula will be in range (x, 10 + x). Note that each execution of sample is performed with *num_iterations* = 1000 setting, which means that algorithm performed 1000 iterations for each sample. Here are the results of the initial testing with 21 samples:

Clauses: [('x3', '!x1'), ('!x1', '!x3'), ('x1', '!x2'), ('!x3', 'x2'), ('x1', 'x3'), ('x2', 'x3'), ('x2', '!x3'), ('x3', '!x2'), ('!x1', 'x3'), ('x2', 'x1')]

Variables: x1 x2 x3

Best Assignments:

1 1 1

0 1 1

Best Satisfaction: 9

Clauses: [('!x3', '!x1'), ('x1', '!x4'), ('!x2', 'x3'), ('!x3', '!x5'), ('!x6', '!x1'), ('!x2', '!x4'), ('x6', '!x4')]

Variables: x1 x2 x3 x4 x5 x6

Best Assignments:

0 0 0 0 1 1

0 0 1 0 0 0

0 1 1 0 0 1

0 0 0 0 0 0

1 0 0 0 0 0

0 0 0 0 1 0

0 1 1 0 0 0

0 0 1 0 0 1

0 0 0 0 0 1

1 0 0 0 1 0

Best Satisfaction: 7

Clauses: [('x1', '!x2'), ('!x4', '!x2'), ('!x3', 'x1'), ('!x1', '!x3'), ('x3', 'x4'), ('!x2', '!x1'), ('!x2', '!x3'), ('!x4', '!x2')]

Variables: x1 x2 x3 x4

Best Assignments:

0 0 0 1

1 0 0 1

Best Satisfaction: 8

Clauses: [('!x5', '!x2'), ('x4', '!x7'), ('!x4', '!x7'), ('x5', '!x6'), ('x5', 'x7'), ('!x6', '!x5'), ('!x4', 'x6')]

Variables: x1 x2 x3 x4 x5 x6 x7

Best Assignments:

0 0 1 0 1 0 0

1 0 1 0 1 0 0

1 0 0 0 1 0 0

0 0 0 0 1 0 0

Best Satisfaction: 7

Clauses: [('x2', 'x4'), ('x6', 'x1'), ('!x3', 'x4'), ('x4', 'x3'), ('x1', 'x5'), ('!x2', 'x5'), ('!x4', 'x2'), ('x3', '!x4'), ('x6', 'x5'), ('!x5', '!x2'), ('!x1', 'x6')]

Variables: x1 x2 x3 x4 x5 x6

Best Assignments:

0 0 1 1 1 1

1 1 1 1 0 1

1 0 1 1 1 1

1 0 1 1 0 1

1 1 1 1 1 1

0 1 1 1 1 1

Best Satisfaction: 10

Clauses: [('!x8', 'x3'), ('!x1', 'x8'), ('!x7', 'x4'), ('x1', '!x3'), ('x1', '!x3'), ('!x6', '!x3'), ('!x7', 'x8'), ('x2', '!x6'), ('!x7', '!x2'), ('!x5', '!x3'), ('x8', 'x2'), ('x3', 'x4')]

Variables: x1 x2 x3 x4 x5 x6 x7 x8

Best Assignments:

1 0 1 0 0 0 0 1

1 0 1 1 0 0 1 1

0 1 0 1 1 0 0 0

0 1 0 1 1 1 0 0

0 1 0 1 0 0 0 0

0 1 0 1 0 1 0 0

1 0 1 1 0 0 0 1

1 1 1 1 0 0 0 1

1 1 1 0 0 0 0 1

Best Satisfaction: 12

Clauses: [('x1', '!x2'), ('!x1', 'x2'), ('x3', '!x1'), ('!x1', 'x3'), ('!x3', '!x1'), ('x1', 'x2')]

Variables: x1 x2 x3

Best Assignments:

0 1 0

0 0 0

0 0 1

1 1 1

0 1 1

Best Satisfaction: 5

Clauses: [('x6', '!x3'), ('x9', '!x1'), ('x6', '!x7'), ('!x3', '!x1'), ('x6', 'x5'), ('!x2', 'x3'), ('x6', 'x8'), ('x6', '!x8'), ('x2', 'x7'), ('x1', 'x8'), ('!x3', 'x4'), ('x1', '!x4')]

Variables: x1 x2 x3 x4 x5 x6 x7 x8 x9

Best Assignments:

1 0 0 0 1 1 1 0 1

1 0 0 1 0 1 1 0 1

1 0 0 1 0 1 1 1 1

0 0 0 0 1 1 1 1 0

1 0 0 1 1 1 1 1 1

0 0 0 0 0 1 1 1 1

1 0 0 0 0 1 1 0 1

1 0 0 0 1 1 1 1 1

1 0 0 0 0 1 1 1 1

Best Satisfaction: 12

Clauses: [('!x4', 'x5'), ('!x5', 'x1'), ('x3', '!x4'), ('!x8', '!x3'), ('x2', '!x4')]

Variables: x1 x2 x3 x4 x5 x6 x7 x8

Best Assignments:

1 1 0 0 1 1 1 1

1 0 0 0 0 0 1 1

0 0 1 0 0 0 0 0

1 1 1 0 1 1 1 0

1 1 0 0 0 0 0 1

1 1 0 0 0 0 1 0

1 1 1 0 1 0 1 0

1 0 0 0 1 0 1 0

1 1 0 0 0 1 1 0

1 0 1 0 0 1 0 0

1 0 0 0 0 1 1 1

1 1 0 0 0 1 0 1

1 1 1 1 1 1 0 0

0 1 1 0 0 0 0 0

1 1 1 0 0 0 1 0

1 0 0 0 1 1 1 0

1 1 1 1 1 0 0 0

1 1 0 0 1 0 0 0

1 0 1 0 1 0 1 0

0 1 0 0 0 0 0 0

0 0 1 0 0 1 1 0

0 0 0 0 0 1 0 0

0 1 0 0 0 1 0 0

1 0 1 0 0 0 0 0

1 1 1 0 0 1 1 0

1 0 0 0 1 1 0 1

0 1 1 0 0 1 0 0

1 0 1 0 1 1 1 0

1 1 0 0 1 1 1 0

1 0 0 0 0 0 1 0

1 1 0 0 0 0 0 0

1 0 0 0 1 0 0 1

1 1 0 0 1 0 1 1

1 1 1 0 1 0 0 0

1 1 0 0 0 1 0 0

0 1 0 0 0 0 1 1

1 0 0 0 0 1 1 0

1 1 0 0 1 1 0 1

1 0 0 0 0 0 0 1

1 1 1 0 1 1 0 0

0 1 0 0 0 1 1 1

0 0 1 0 0 0 1 0

1 1 1 0 0 1 0 0

1 0 1 0 0 1 1 0

1 1 1 1 1 0 1 0

1 0 0 0 0 1 0 1

0 0 0 0 0 0 1 1

0 0 0 0 0 1 1 1

1 1 1 1 1 1 1 0

1 0 0 0 1 1 0 0

1 1 0 0 0 0 1 1

0 1 1 0 0 0 1 0

1 0 1 0 1 0 0 0

1 0 0 0 1 0 0 0

1 1 0 0 1 0 1 0

0 0 1 0 0 1 0 0

0 1 0 0 0 0 1 0

1 0 0 0 1 0 1 1

1 1 0 0 0 1 1 1

1 0 1 0 1 1 0 0

1 0 1 0 0 0 1 0

0 1 1 0 0 1 1 0

1 1 1 0 0 0 0 0

0 1 0 0 0 1 1 0

1 1 0 0 1 1 0 0

1 0 0 0 0 0 0 0

1 0 0 0 1 1 1 1

0 0 0 0 0 0 1 0

1 0 0 0 0 1 0 0

1 1 0 0 1 0 0 1

0 0 0 0 0 0 0 1

0 1 0 0 0 0 0 1

0 0 0 0 0 1 0 1

0 1 0 0 0 1 0 1

0 0 0 0 0 1 1 0

Best Satisfaction: 5

Clauses: [('!x8', '!x1'), ('x2', 'x3'), ('x2', '!x6'), ('x5', '!x8'), ('!x5', 'x1'), ('!x3', 'x1'), ('x2', '!x6'), ('!x1', 'x7'), ('x7', '!x5'), ('!x7', '!x6')]

Variables: x1 x2 x3 x4 x5 x6 x7 x8

Best Assignments:

1 1 1 1 0 0 1 0

0 1 0 0 0 0 1 0

1 1 0 1 1 0 1 0

1 0 1 1 1 0 1 0

0 1 0 0 0 1 0 0

1 0 1 1 0 0 1 0

1 1 0 0 0 0 1 0

1 0 1 0 0 0 1 0

1 1 1 1 1 0 1 0

1 1 1 0 1 0 1 0

1 1 0 0 1 0 1 0

1 1 0 1 0 0 1 0

0 1 0 1 0 1 0 0

0 1 0 1 0 0 0 0

0 1 0 1 0 0 1 0

1 0 1 0 1 0 1 0

0 1 0 0 0 0 0 0

Best Satisfaction: 10

Clauses: [('!x4', '!x1'), ('!x4', '!x5'), ('x5', 'x3'), ('x2', 'x5'), ('x4', 'x1'), ('!x1', '!x2'), ('x1', '!x3'), ('x4', '!x5'), ('!x5', '!x2'), ('x5', '!x3')]

Variables: x1 x2 x3 x4 x5

Best Assignments:

0 0 0 1 1

1 0 1 0 1

0 1 0 1 0

1 0 0 0 1

Best Satisfaction: 9

Clauses: [('!x7', 'x2'), ('x3', '!x5'), ('!x2', 'x3'), ('x6', 'x7'), ('x5', '!x4'), ('x5', '!x4'), ('x2', '!x8'), ('x7', 'x3'), ('x7', 'x5'), ('x8', 'x3'), ('x5', '!x6')]

Variables: x1 x2 x3 x4 x5 x6 x7 x8

Best Assignments:

1 1 1 0 1 1 1 0

1 1 1 0 1 0 1 0

0 1 1 1 1 1 0 1

0 1 1 0 1 1 1 0

1 1 1 1 1 1 0 0

1 1 1 0 0 0 1 0

0 1 1 0 1 1 0 1

1 1 1 0 1 1 0 1

1 1 1 1 1 0 1 1

1 1 1 1 1 1 1 1

0 0 1 1 1 1 0 0

0 1 1 1 1 1 0 0

1 1 1 0 1 1 0 0

0 1 1 0 1 1 0 0

0 1 1 1 1 0 1 1

0 1 1 1 1 1 1 1

1 1 1 1 1 0 1 0

0 1 1 0 1 0 1 1

0 0 1 0 1 1 0 0

1 1 1 1 1 1 1 0

0 1 1 0 0 0 1 0

1 1 1 0 1 1 1 1

1 1 1 0 1 0 1 1

1 0 1 0 1 1 0 0

1 1 1 1 1 1 0 1

1 0 1 1 1 1 0 0

0 1 1 0 1 1 1 1

0 1 1 1 1 0 1 0

1 1 1 0 0 0 1 1

0 1 1 1 1 1 1 0

0 1 1 0 1 0 1 0

Best Satisfaction: 11

Clauses: [('x1', 'x3'), ('x6', 'x3'), ('x1', 'x8'), ('x2', '!x5'), ('!x7', '!x3')]

Variables: x1 x2 x3 x4 x5 x6 x7 x8

Best Assignments:

1 1 0 0 1 1 1 1

1 0 0 1 0 1 1 1

1 0 1 1 0 0 0 0

1 1 1 0 1 0 0 1

0 1 1 1 1 1 0 1

1 1 1 1 0 1 0 1

1 1 0 0 0 1 1 0

1 0 1 1 0 1 0 1

1 1 0 1 0 1 1 1

1 0 0 0 0 1 1 1

1 1 0 0 0 1 0 1

1 1 1 1 1 1 0 0

1 1 1 0 1 1 0 1

0 1 1 0 1 1 0 1

0 0 1 1 0 1 0 1

1 1 1 1 1 0 0 0

1 1 1 0 0 1 0 1

1 0 1 0 0 0 0 0

1 1 0 1 1 1 1 0

1 0 0 1 0 1 0 1

1 1 0 0 1 1 1 0

1 0 0 1 0 1 1 0

1 1 0 1 1 1 0 1

1 1 1 1 0 0 0 0

0 1 1 1 0 1 0 1

1 1 1 1 0 1 0 0

1 0 1 1 0 1 0 0

1 1 0 1 0 1 1 0

1 0 0 0 0 1 1 0

0 0 1 0 0 1 0 1

1 1 1 0 1 0 0 0

1 1 0 0 0 1 0 0

1 1 1 0 1 1 0 0

1 1 1 0 0 0 0 1

1 1 0 0 1 1 0 1

0 1 1 1 1 0 0 1

0 0 1 1 0 0 0 1

1 0 1 0 0 1 0 1

1 1 1 0 0 1 0 0

0 1 1 1 0 0 0 1

1 1 0 1 0 1 0 1

1 0 0 0 0 1 0 1

1 1 0 1 1 1 0 0

0 0 1 0 0 0 0 1

1 0 0 1 0 1 0 0

1 0 1 1 0 0 0 1

1 0 1 0 0 0 0 1

1 1 0 0 0 1 1 1

1 1 1 0 0 0 0 0

1 1 1 1 1 1 0 1

1 1 0 0 1 1 0 0

0 1 1 0 0 0 0 1

1 1 1 1 1 0 0 1

1 1 0 1 0 1 0 0

1 0 0 0 0 1 0 0

0 1 1 0 0 1 0 1

1 0 1 0 0 1 0 0

1 1 1 1 0 0 0 1

1 1 0 1 1 1 1 1

Best Satisfaction: 5

Clauses: [('!x3', '!x1'), ('x1', 'x2'), ('x1', '!x2'), ('!x2', '!x1'), ('x1', 'x2'), ('!x1', '!x3'), ('!x2', '!x1'), ('x1', 'x2'), ('!x2', '!x3'), ('!x3', 'x1')]

Variables: x1 x2 x3

Best Assignments:

1 0 0

Best Satisfaction: 10

Clauses: [('x6', 'x5'), ('!x5', 'x1'), ('x1', '!x3')]

Variables: x1 x2 x3 x4 x5 x6

Best Assignments:

1 1 1 1 0 1

1 1 1 1 1 0

1 1 0 0 1 0

1 0 0 1 1 1

1 0 1 0 1 1

1 1 0 1 1 0

0 1 0 0 0 1

1 1 0 1 0 1

1 1 1 0 1 1

1 1 0 0 0 1

1 0 0 0 1 1

1 0 1 1 1 1

1 0 1 0 1 0

1 0 0 1 0 1

1 0 0 1 1 0

0 0 0 1 0 1

1 0 1 0 0 1

1 1 1 0 0 1

1 1 1 0 1 0

1 0 1 1 0 1

1 1 1 1 1 1

1 0 1 1 1 0

1 0 0 0 0 1

1 0 0 0 1 0

1 1 0 0 1 1

1 1 0 1 1 1

0 1 0 1 0 1

0 0 0 0 0 1

Best Satisfaction: 3

Clauses: [('!x1', '!x4'), ('!x4', '!x3'), ('x5', '!x4')]

Variables: x1 x2 x3 x4 x5

Best Assignments:

1 0 1 0 0

1 1 0 0 0

0 0 1 0 1

0 1 0 1 1

1 1 1 0 0

0 1 0 0 1

0 0 1 0 0

1 0 0 0 1

0 0 0 0 1

0 0 0 1 1

0 1 0 0 0

0 1 1 0 1

0 0 0 0 0

1 1 1 0 1

1 0 0 0 0

0 1 1 0 0

1 1 0 0 1

1 0 1 0 1

Best Satisfaction: 3

Clauses: [('!x1', '!x2'), ('!x3', '!x1'), ('!x2', 'x3'), ('x3', 'x2'), ('!x2', '!x1'), ('!x3', 'x2'), ('x3', '!x2'), ('x3', 'x2')]

Variables: x1 x2 x3

Best Assignments:

0 1 1

Best Satisfaction: 8

Clauses: [('x3', '!x8'), ('x7', '!x2'), ('x4', 'x2'), ('x5', 'x4'), ('!x3', 'x6')]

Variables: x1 x2 x3 x4 x5 x6 x7 x8

Best Assignments:

1 1 1 0 1 1 1 0

1 0 1 1 1 1 1 1

1 1 1 1 0 1 1 0

0 0 1 1 1 1 0 1

1 0 0 1 0 0 1 0

1 0 0 1 1 1 1 0

1 0 1 1 0 1 0 1

0 1 1 0 1 1 1 0

0 0 0 1 0 1 0 0

0 0 1 1 0 1 1 0

0 1 1 1 0 1 1 0

0 0 1 1 0 1 0 1

0 1 0 0 1 1 1 0

0 1 0 1 0 1 1 0

0 0 0 1 1 0 1 0

1 1 0 1 1 1 1 0

1 1 0 0 1 1 1 0

1 0 0 1 0 1 1 0

1 1 1 1 1 1 1 1

0 0 0 1 1 1 1 0

0 1 0 1 1 0 1 0

0 0 1 1 1 1 0 0

0 0 0 1 0 0 0 0

1 0 1 1 1 1 1 0

1 1 0 1 0 0 1 0

1 0 1 1 0 1 0 0

1 1 0 1 0 1 1 0

1 0 0 1 1 0 0 0

0 1 0 1 0 0 1 0

0 0 1 1 0 1 0 0

1 0 1 1 1 1 0 1

0 0 1 1 1 1 1 1

1 0 0 1 0 0 0 0

1 0 0 1 1 1 0 0

0 1 0 0 1 0 1 0

1 0 1 1 0 1 1 1

1 1 1 1 1 1 1 0

0 0 0 1 1 1 0 0

1 0 0 1 0 1 0 0

0 0 0 1 0 1 1 0

1 1 1 0 1 1 1 1

1 1 0 0 1 0 1 0

1 1 1 1 0 1 1 1

1 1 0 1 1 0 1 0

1 0 1 1 1 1 0 0

0 1 1 0 1 1 1 1

0 0 1 1 0 1 1 1

0 1 1 1 0 1 1 1

0 0 1 1 1 1 1 0

0 0 0 1 0 0 1 0

0 1 1 1 1 1 1 0

0 0 0 1 1 0 0 0

1 0 1 1 0 1 1 0

0 1 0 1 1 1 1 0

1 0 0 1 1 0 1 0

Best Satisfaction: 5

Clauses: [('x4', '!x5'), ('x3', '!x2'), ('x1', '!x2'), ('x2', '!x5'), ('x3', '!x2'), ('x5', 'x4'), ('!x3', 'x4'), ('!x6', 'x4'), ('!x5', '!x4'), ('!x4', '!x5'), ('x2', '!x3'), ('!x1', 'x5')]

Variables: x1 x2 x3 x4 x5 x6

Best Assignments:

0 0 0 1 0 0

0 0 0 1 0 1

Best Satisfaction: 12

Clauses: [('!x2', '!x1'), ('!x2', '!x1'), ('x1', 'x2'), ('!x1', 'x2')]

Variables: x1 x2

Best Assignments:

0 1

Best Satisfaction: 4

Clauses: [('x2', 'x3'), ('!x2', '!x1'), ('x1', 'x3'), ('x1', 'x4'), ('x4', '!x1'), ('x4', 'x2'), ('x2', '!x1'), ('x1', '!x4'), ('x3', 'x1'), ('!x4', 'x2')]

Variables: x1 x2 x3 x4

Best Assignments:

1 1 0 1

0 1 1 0

0 1 1 1

1 1 1 1

Best Satisfaction: 9

6. Experimental Analysis of The Performance (Performance Testing)

To analyse the performance of the implementation of the algorithm experimentally, we can perform multiple measurements on a fixed formula and fixed number of iterations because for a certain size of input, taking only a single measurement for the running time will not give a confident measurement. For measurement, a random formula containing 6 variables and 10 clauses was generated using our sample generator. The formula is as follows.

[("x3", "x2"), ("x1", "!x4"), ("x6", "x5"), ("x5", "x1"), ("x2", "!x3"), ("x4", "!x2"), ("!x6", "x2"), ("x3", "!x6"), ("!x3", "x4"), ("x6", "!x5")]

This formula is measured 20 times with 1000 number of iterations each time. It achieved a mean runtime of 0.0105 seconds with standard deviation of 0.0012 seconds. With 90% confidence interval, we get [0.0102, 0.0109] seconds as a result. The resulting graph is provided below. From the fitted line for the running time in this graph, we can get the running time expression as follows.

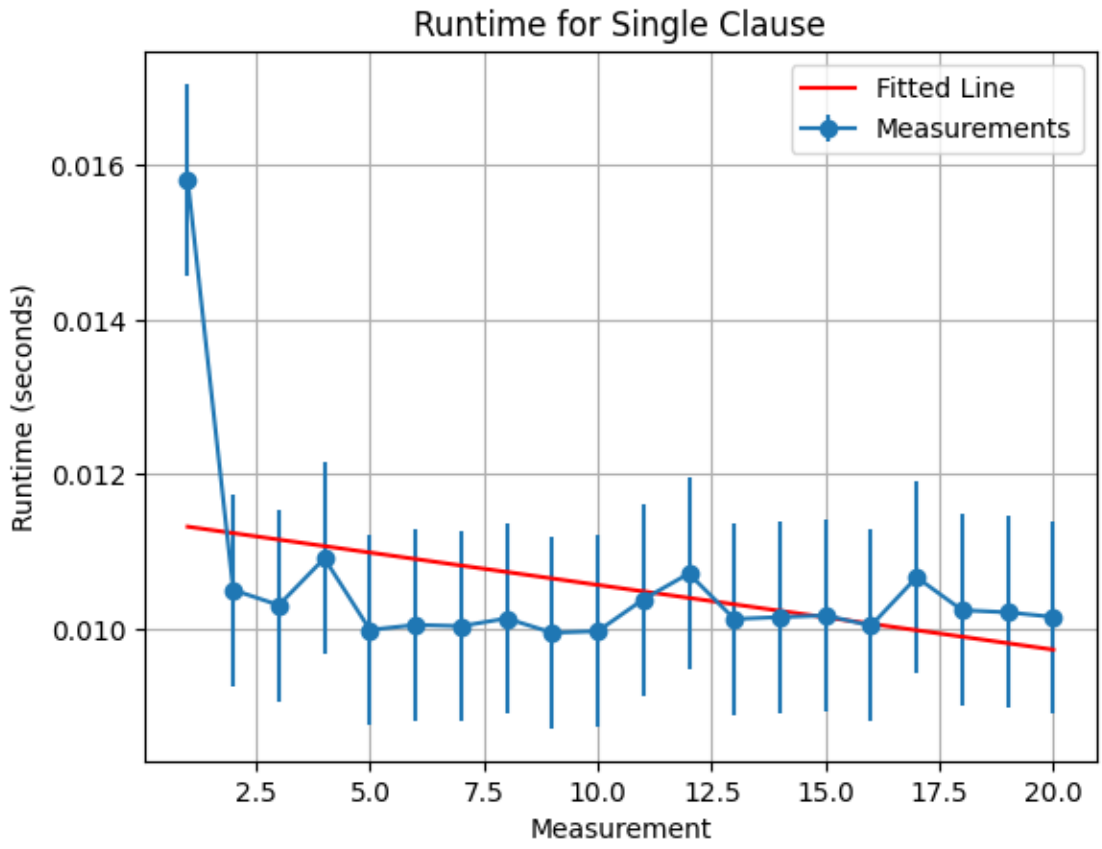
Running time expression: -0.0001 * x + 0.0114

Note that there can be outliers in the graph, for example first measurement. This happens because of the nature of our algorithm. As our algorithm uses a randomized approach, it does not guarantee to provide correct solutions every time and on some occasions, it may take longer to come up with a solution.

To gain a better understanding of how this randomized approach heuristic algorithm performs compared to our brute force algorithm, I've also measured the runtime of the brute force algorithm with using the same formula. The brute force algorithm took 0.00190266 seconds using the same clause provided above. Do not forget that our results from heuristic algorithm is measured by considering average time of 20 measurements and 1000 iteration in each measurement. When we measure our heuristic algorithm one time with number of iterations set to 1, it will performs similarly compared to the brute force algorithm, but it won't produce correct result. For example, when I run the heuristic algorithm once, with number of iterations equal to 1, it produces the following result:

Best Satisfaction: 7

Runtime: 0.0015020370483398438 seconds



7. Experimental Analysis of the Quality

To analyse the quality of the solutions that our randomized heuristic algorithm produces, we need to consider two important factors that have a significant impact on the algorithm from many aspects: Sample size and number of iterations. I've generated 6 samples randomly, which contains 11, 21, 41, 61, 81 and 121 clauses respectively, to observe the impact of the sample size to the quality of the solutions that heuristic algorithm outputs. I've tested all these samples separately on this algorithm with number of iterations fixed to 1000 in each measurement. Results were not surprising, the sample size of the input did not have any effects on the solution quality, all the solutions were correct. I've also performed some testing to check whether number of clauses effects the quality of the results. I've tested 7 formulas which are randomly generated containing 2, 4, 8, 16, 32, 64 and 128 clauses respectively. All formulas had 4 variables and number of iterations was fixed to 1000. All the results were turned out to be correct. Next, I've checked whether the number of variables has an impact on the quality of the solutions or not. To perform this test, I've created 5 formulas containing 6, 8, 10, 12, and 16 variables. All formulas had 8 clauses and number of iterations was

fixed to 1000. Maximum number of clauses satisfied results were correct, but the heuristic algorithm did not display all best possible truth assignments in some formulas. I assume that happened because of the number of iterations were limited and the fact that algorithm makes the truth assignments randomly.

To observe the impact of the number of iterations on the correctness of the solutions, I've created a sample randomly which contains 21 formulas. I've executed the heuristic algorithm using this sample many times, with different number of iterations each time. The number of the iterations I've tried was: 2000, 1000, 500, 250, 100, 50, 25, and 10. I've created a table to represent the result of each measurement. Note that the column for "Corresponding Truth Values for the Variables" is created considering the missing assignments as wrong. For example, if a formula does not contain all possible truth assignments which satisfies maximum number of clauses, it is considered as wrong. This column shows true assignments out of all assignments. Table is provided below.

Number of Iterations	Maximum Number of Clauses Satisfied	Corresponding Truth Values for the Variables
2000	21/21	21/21
1000	21/21	20/21
500	21/21	13/21
250	21/21	13/21
100	20/21	10/21
50	18/21	8/21
25	20/21	7/21

From this table, we can see that in every different number of iterations performed, algorithm can find maximum number of clauses satisfied almost correctly for each formula. Whereas we can observe that it cannot perform equally better at finding the corresponding truth values for the variables in each formula. When the number of iterations goes lower than 1000, there is a significant drop. This result occurs because of the testing method that I've followed and the nature of the algorithm. As I count results with missing assignments wrong, some results were directly considered as false even though the assignments that they contain were true, but incomplete. Also, as

algorithm assigns values to variables randomly and the number of iterations is limited, we can see such results. In some cases where the sample size is big and number of iterations were low compared to the sample size, algorithm may produce incomplete results due to the insufficient number of iterations required to output all the possible truth assignments that satisfies maximum number of clauses.

8. Experimental Analysis of the Correctness (Functional Testing)

Black box testing on this randomized heuristic algorithm was performed using previous randomly generated samples containing 11, 21, 41, 61, 81 and 121 formulas. Testing was performed with the number of iterations fixed to 1000 for each sample. Results of this heuristic algorithm were compared to the results that the brute force algorithm produced, and no problems were found. For all samples, algorithm produced maximum number of clauses satisfied and all possible best truth assignments correctly. From the previous section, we know that the correctness of this algorithm is directly proportional with the number of iterations performed. In a situation that these samples were executed by the heuristic algorithm which has smaller number of iterations to be performed, algorithm might probably output incorrect/incomplete results. As we fixed the number of iterations to be performed to 1000 for testing purposes, we do not have to worry about this situation. As a result of black box testing, we can say that our randomized heuristic algorithm works correctly and does not have any coding errors.

9. Discussion

According to the results of the experiments that I've performed on this heuristic algorithm, it produces correct results, and I didn't encounter any inconsistencies between theoretical analysis and experimental analysis. One thing which is important for the correctness of the algorithm is that its correctness is heavily dependent on the number of iterations performed. Due to the random nature of the algorithm, it may not guarantee an optimal solution in every run. For example, as we can see from the section 7, in the case that a smaller number of iterations performed, algorithm can produce incomplete/incorrect results. However, the more iterations the algorithm performs, the higher the probability of finding the best assignments. Still, it is a better alternative than the brute force algorithm as it consumes much less time and resources compared to the brute force approach. I believe that with further testing, optimizations on the number of

iterations to be performed can be made depending on the sample used. Hence, the overall performance of the algorithm can be improved.

Submission

On SUCourse, for each group, only ONE person will submit the report.

For Progress Reports, we expect a report as PDF file. Your filename must be in the following format:

`CS301_Project_Progress_Report_Group_XXX.pdf`

where `XXX` is your group number.

For Final Reports, we expect a report as a PDF file, and also the codes and the data produced during the experiments. Please submit a single zip file that contains all your project files. Your zip file name must be in the following format:

`CS301_Project_Final_Report_Group_XXX.zip`

where `XXX` is your group number. In this zip file, please make sure that your final report is named as follows:

`CS301_Project_Final_Report_Group_XXX.pdf`

where `XXX` is your group number.