NP-Hard Presentation

Max 3-SAT

Caleb Collins, James Tremblay, Charles Kimbrough

JMU JAMES MADISON UNIVERSITY

Exact Solution

James Tremblay

Problem Description & applications

- Given a collection of 3-literal clauses

- Find the variable assignment that maximizes the number of satisfied clauses

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_3 \vee x_4)$$

- Circuit design – maximizing satisfied constraints
- Scheduling – handle conflicting requirements
- Resource allocation – choosing assignments that satisfy the most conditions.

Inputs / Outputs

n (variables/x's) m (clauses / lines)
Numbers represent which x variable is
Negative numbers represent negative values

4 5
-1 -2 -3
-3 -2 -4
4 -1 2
2 -4 -3
-1 -2 3

$$(-x_1 \vee -x_2 \vee -x_3) \wedge$$
$$(-x_3 \vee -x_2 \vee -x_4) \wedge$$
$$(x_4 \vee -x_1 \vee x_2) \wedge$$
$$(x_2 \vee -x_4 \vee -x_3) \wedge$$
$$(-x_1 \vee -x_2 \vee x_3)$$

Solution that satisfies the most clauses

First line is number of clauses satisfied

5
1 F
2 F
3 F
4 F

$$x_1 = False$$
$$x_2 = False$$
$$x_3 = False$$
$$x_4 = False$$

Polynomial Modification

If the clauses only had 2 variables it would be solvable in linear time

In 2-SAT, implications form a graph with no branching
Just follow arrows until you hit a contradiction

$$(-x_1 \vee -x_2) \wedge$$
$$(x_2 \vee x_3)$$

In 3-SAT, implications creates a decision tree
with splits and exploring both branches takes time

Reduction, Intro to SAT

Given a collection of literal clauses

Any number of literals per line

Trying to satisfy every clause

$$(-x_1 \lor -x_2) \land$$
$$(x_2 \lor -x_4 \lor -x_3) \land$$
$$(x_5)$$

Outputs: if all clauses are satisfiable or not

Reduction SAT -> Max 3Sat

No changes to the input

For some number of clauses, k, we ask our SAT solver if it can satisfy at least k clauses.

Use binary search from 0 to m to find the largest k (log(m))

k is the Max-3SAT solution

Reduction   Max 3SAT -> SAT

Clauses with less than 3 literals
add a redundant variable

$$(-x_1 \vee x_2) \rightarrow (-x_1 \vee x_2 \vee x_2)$$

Clauses with more than 3 literals are broken up into multiple clauses

$$(x_1 \vee x_2 \vee x_3 \vee x_4) \rightarrow (x_1 \vee x_2 \vee b) \wedge (-b \vee x_3 \vee x_4)$$

Run Max 3SAT solver and take result x

(x == #clauses) answer to SAT

Program

Checks if a clause satisfied

```
def clause_sat(clause, assignment):
    a, b, c = clause
    return (
        (a > 0 and assignment[a-1]) or (a < 0 and not assignment[-a-1]) or
        (b > 0 and assignment[b-1]) or (b < 0 and not assignment[-b-1]) or
        (c > 0 and assignment[c-1]) or (c < 0 and not assignment[-c-1])
    )
```

Loops over all possible assignments and finds the one that satisfies the most clauses

```
for a in itertools.product([False, True], repeat=n):
    val = sum(clause_sat(c, a) for c in clauses)
    if val > best_val:
        best_val = val
        best_assignment = a
```

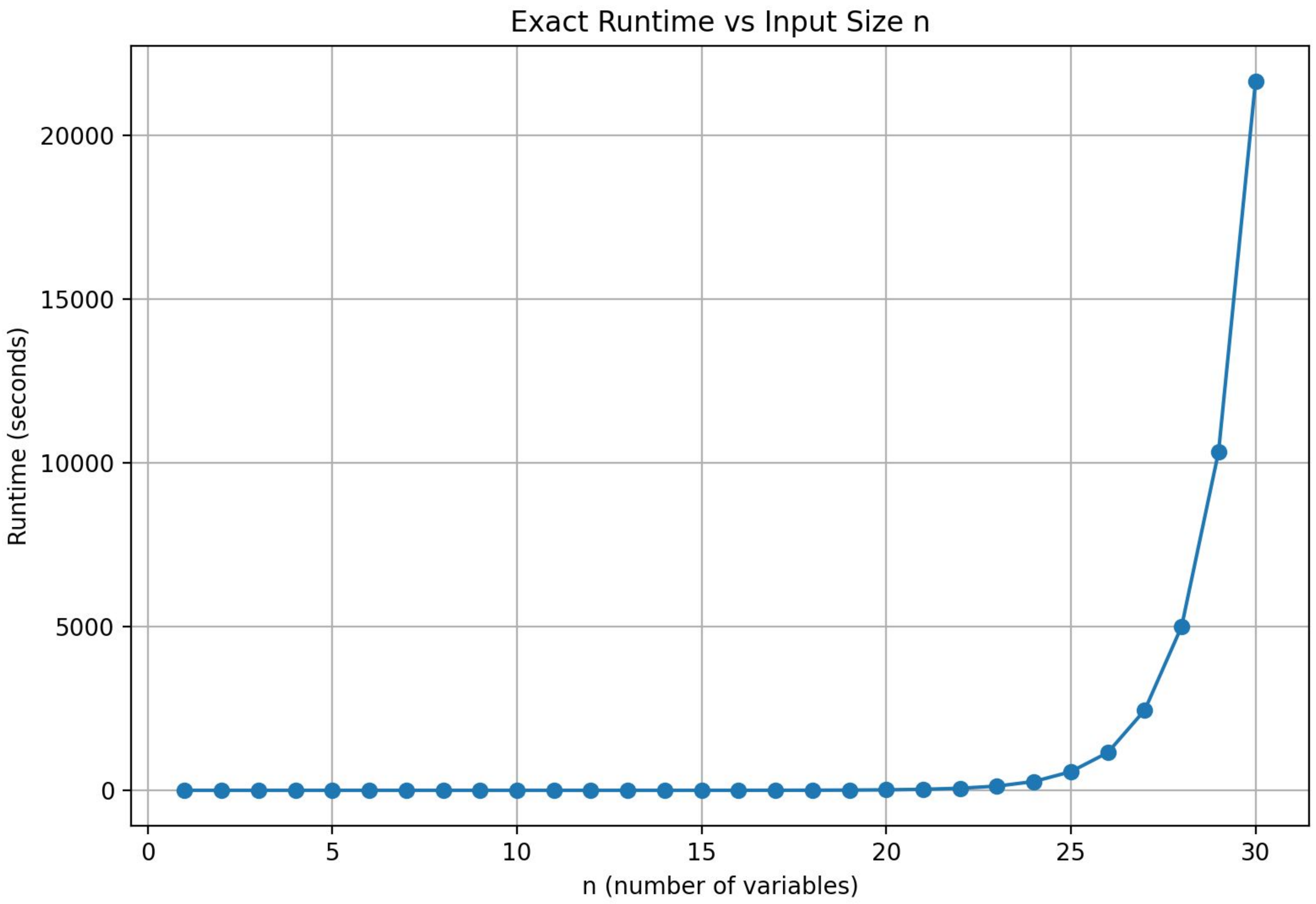
Input Changes affect speed

For any ordering the exact solution goes through the same set of assignments and the same number of clause checks

Changing the input order will not reduce the amount of time spent

The only change to input to make program faster is to reduce the amount of variables which reduces the total possible number of assignments

Plot


Exact Runtime vs Input Size n
Runtime (seconds)
20000
15000
10000
5000
0
0
5
10
15
20
25
30
n (number of variables)

Approximate Solution

Charles Kimbrough

Approach

1. Start With a Random Assignment
    a. Each variable is randomly set to True/False
    b. All clauses are evaluated once
    c. A list for each of the satisfied and unsatisfied clauses is created
2. Repeatedly Fix an Unsatisfied Clause
    a. Every iteration chooses one random unsatisfied clause and tries to fix it using one of two strategies. Randomly selected which one is chosen.

Strategy A: Random Walk

Has a probability of 0.4 to occur

Pick a literal from the unsatisfied clause and flip its variable

This is to escape any local minima by introducing random noise.

Strategy B: Greedy Improvement

Has a probability of 0.6 to occur

Evaluate the gain from flipping each variable in the clause

gain(x) = Δ(number of satisfied clauses)

Pick the variable with the best positive gain, tie-broken randomly

This is to climb uphill by making the single flip that increases satisfaction the most.

Approach Continued

After a certain number of max flips, the algorithm:

- saves its best score so far
- completely restarts with a new random assignment

This is to avoid getting stuck in a hole, and try a new selection

This runs for a certain time before quitting, to satisfy the anytime.

If the best score equals number of clauses, the algorithm quits early.

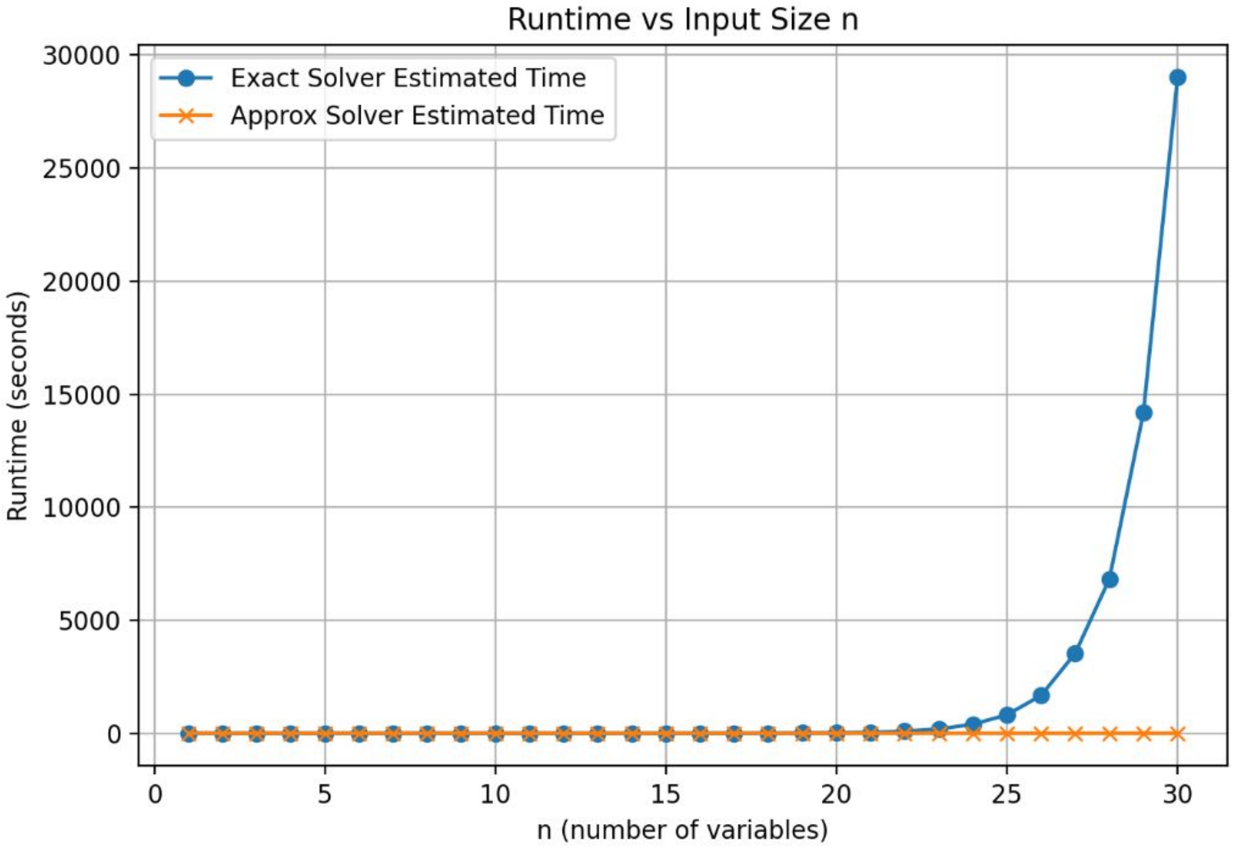
Runtime

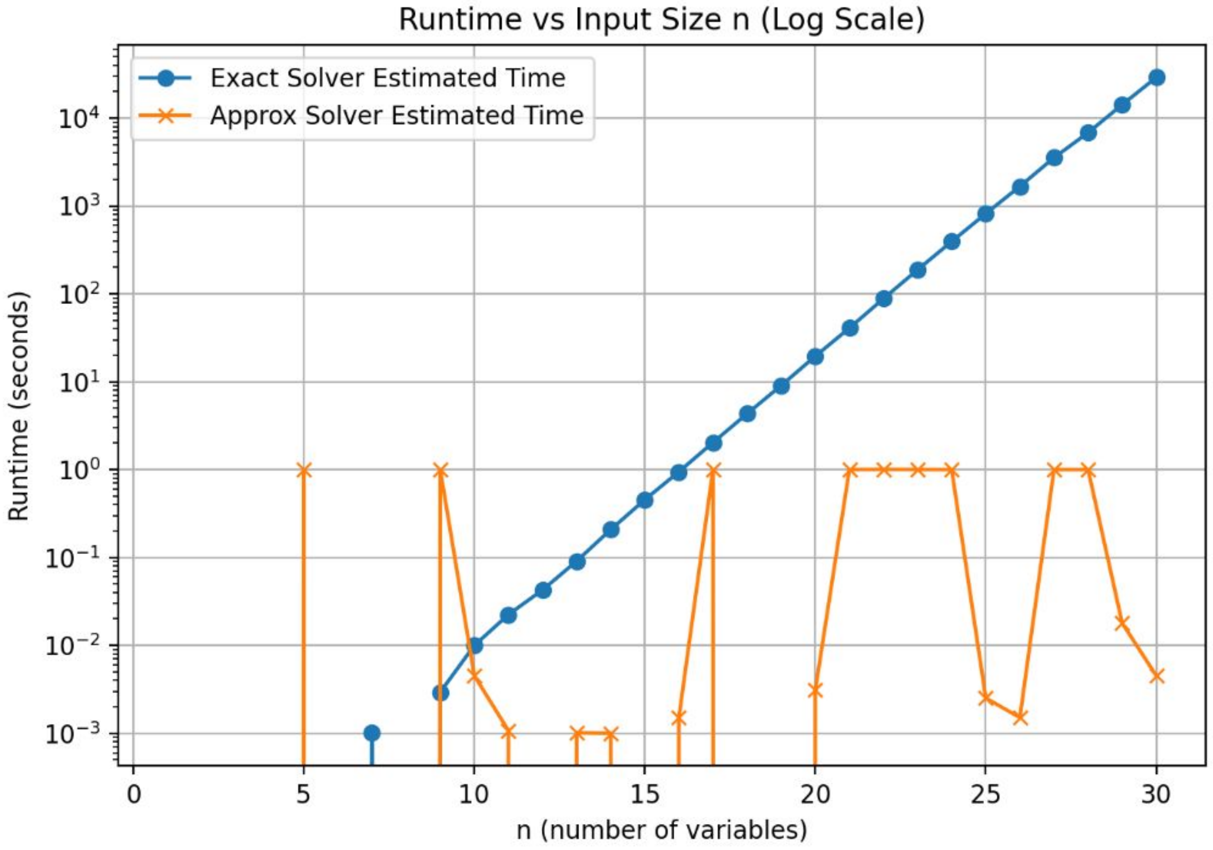
n = number of variables

m = number of clauses

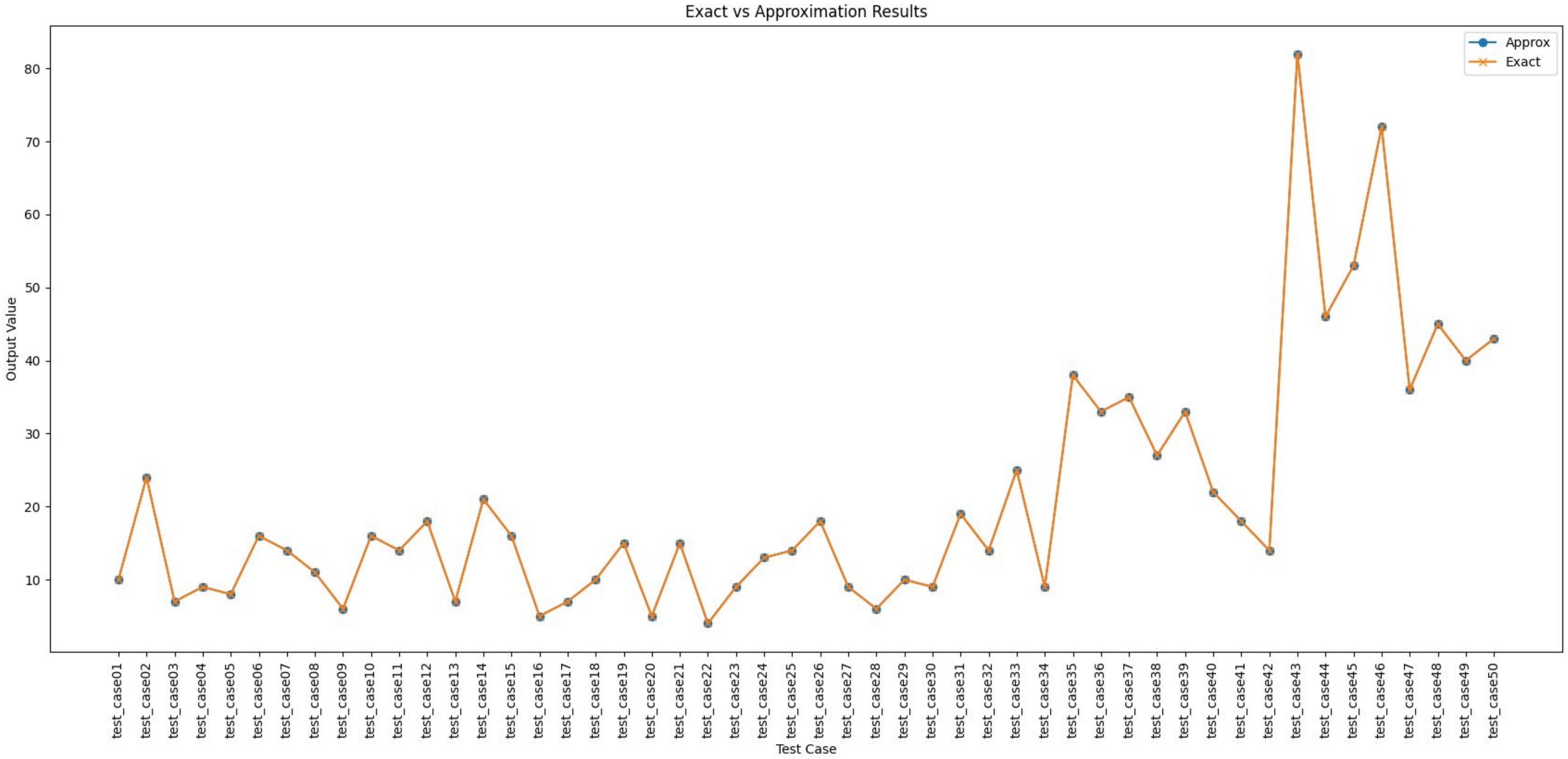
The cost per flip is O(m/n). Since this runs for a constant number for flips, the runtime for a single restart is O(m/n). Since most of time m>>n, it is closer to O(m).

Approximation vs Exact Solution



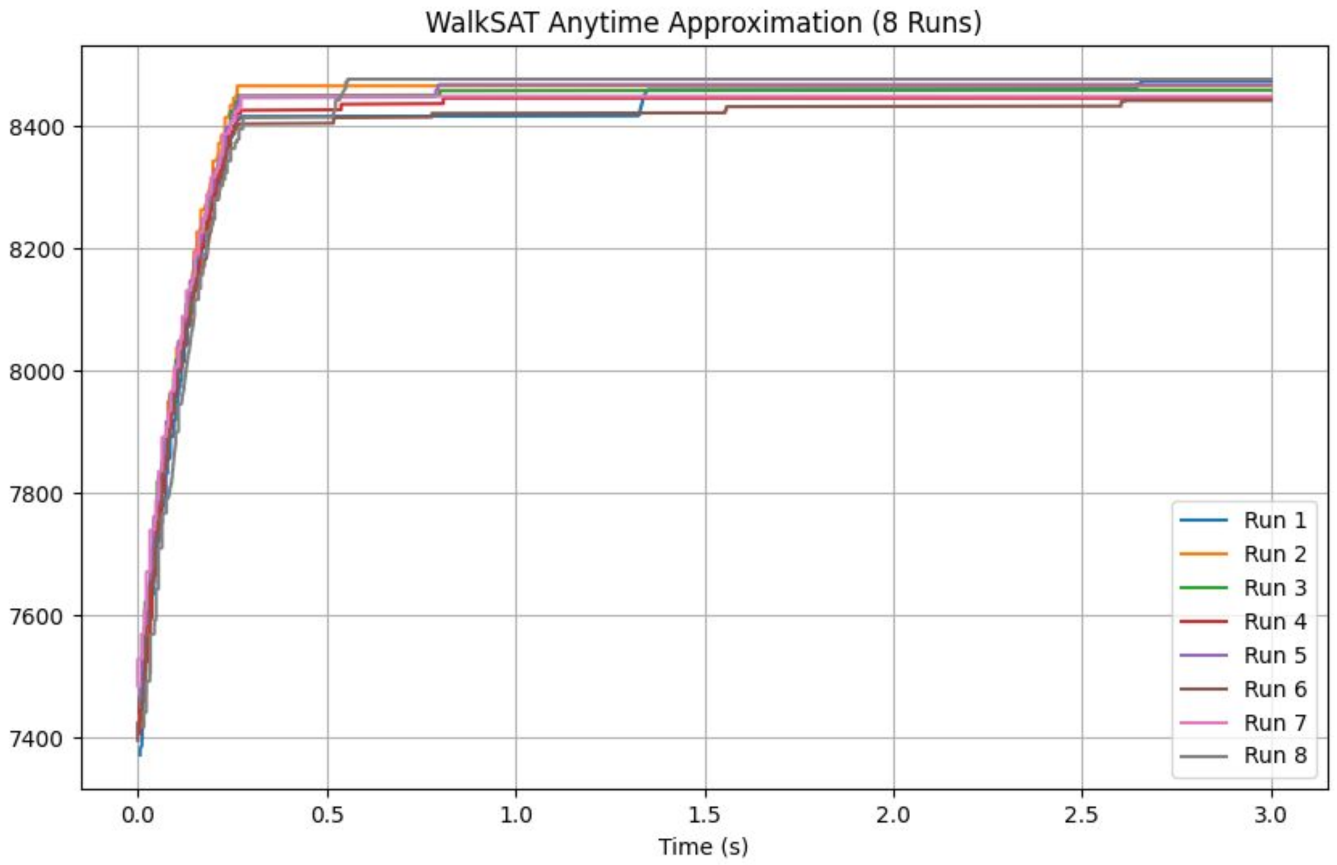


Approximation vs Exact Solution



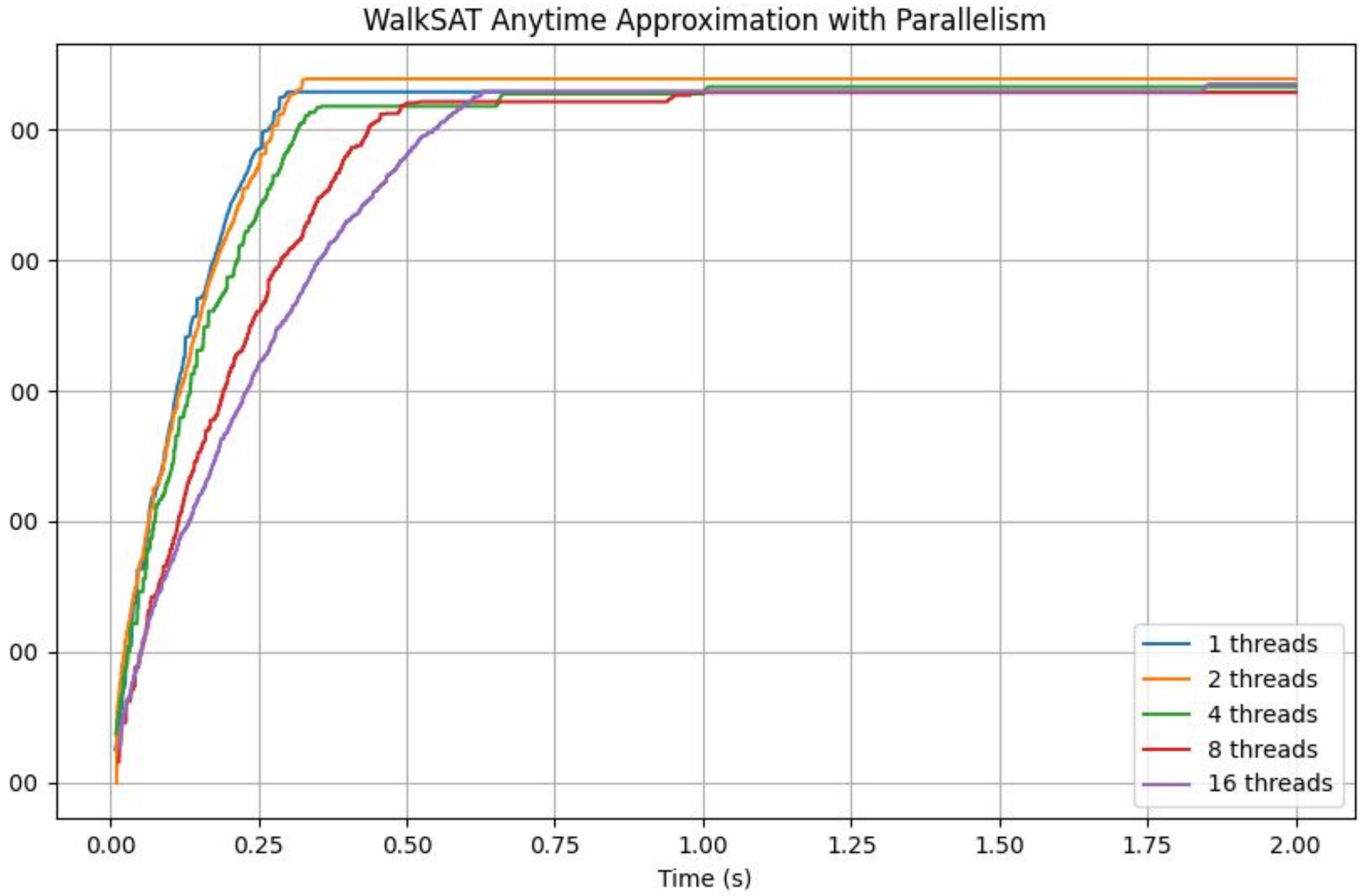
JAMES MADISON UNIVERSITY

Solution Over Runtime



Parallelism

- Adding parallelism, decreased the initial speed of finding a solution.
- There was no consistent improvement in results for different parallelism.

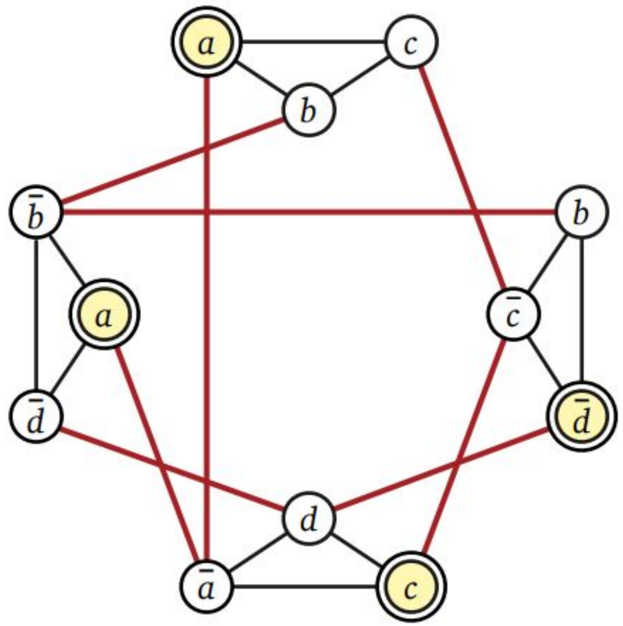


Reduced Solution

Max 3-SAT to Max Independent Set (MIS)

Max 3-SAT Reduction Algorithm

1. Initialize Graph
2. Map Literals => 3c = V
3. Identify Clause Conflicts (△)
4. Identify Logical Conflicts
5. Build Edge List

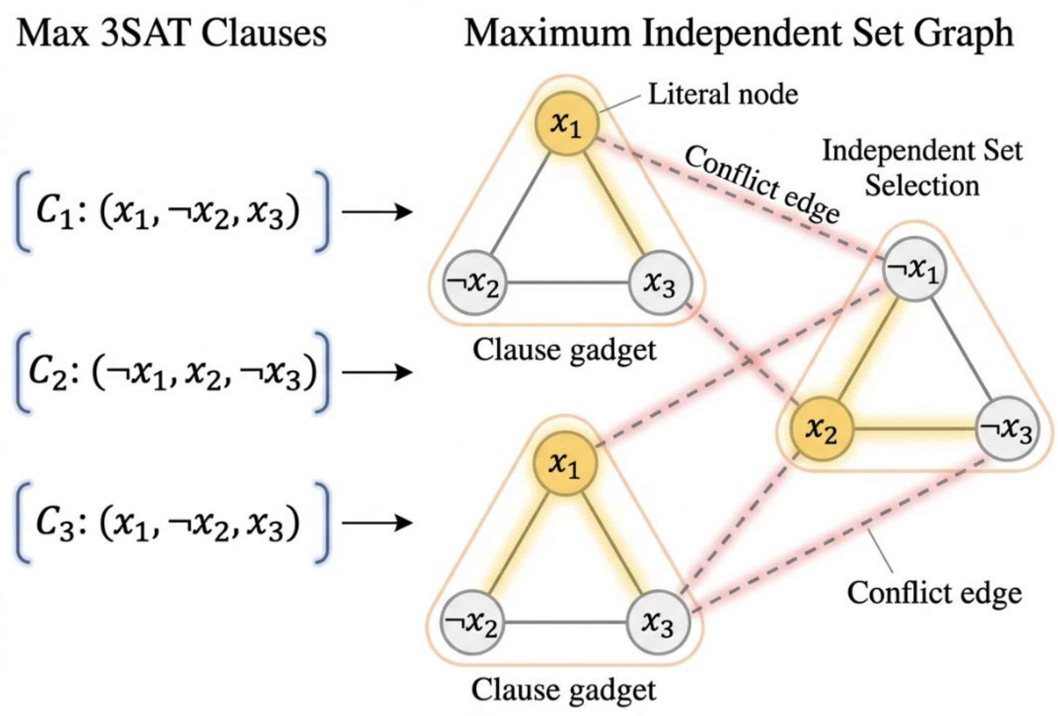


$(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$

The runtime complexity of this reduction is O(c^2), where c is the number of clauses, due to the nested loops for edge construction.

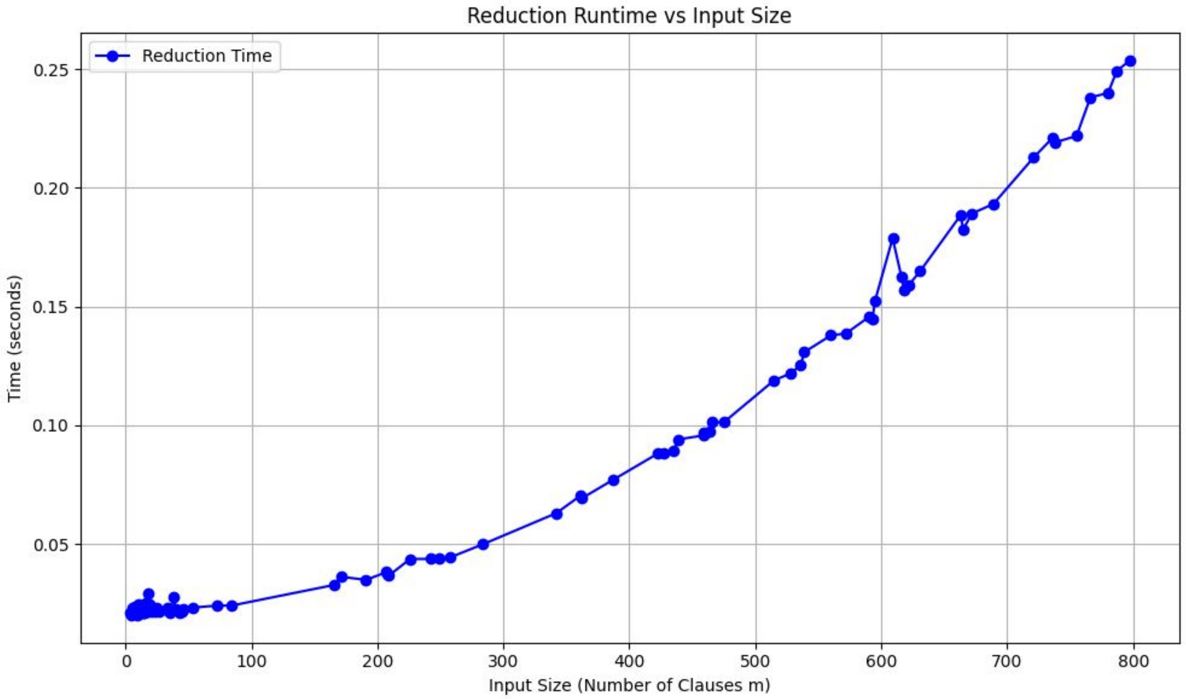
Reduction Explanation

- Clause Gadget (Triangles): Each 3-literal clause forms a triangle. This complete subgraph ensures only one vertex (literal) can be selected per clause.
- Conflict Edges (Dashed): Edges connect every literal x to its negation ¬x. This prevents contradictory literals from existing in the same independent set.
- Core Intuition: Selecting one non-conflicting literal per clause creates an independent set. The size of this set equals the number of satisfiable clauses.



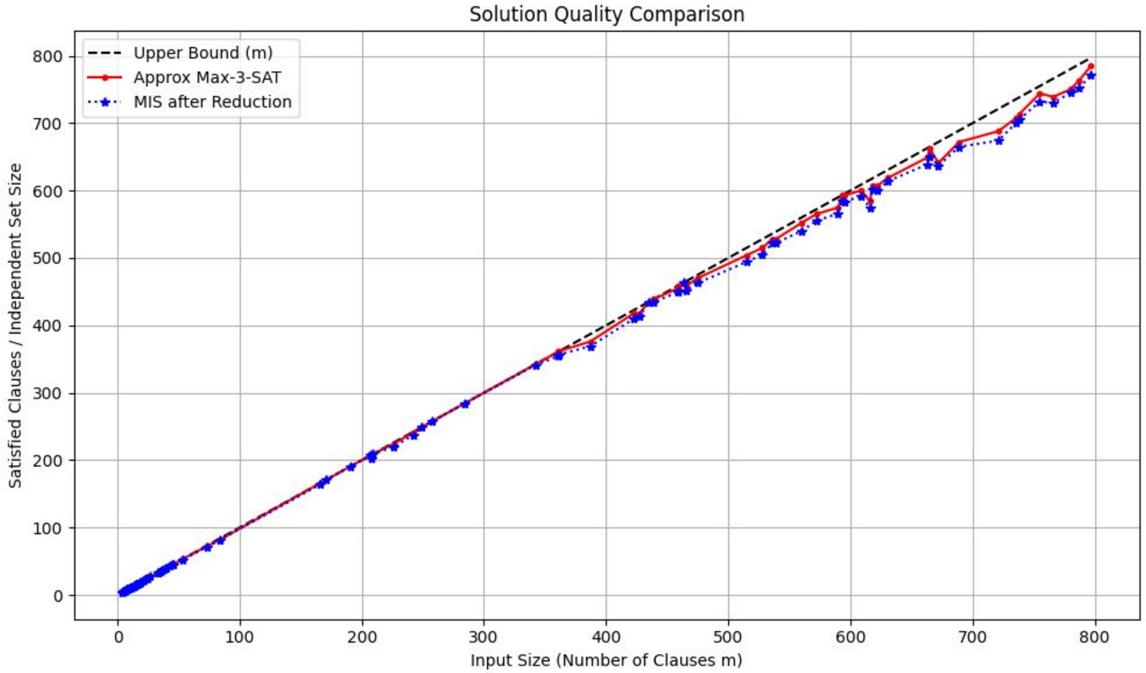
Runtime For Reduction

- The runtime complexity of this reduction is O(c^2), where c is the number of clauses, due to the nested loops for edge construction. The outer loop is clause construction and inner is checking for logical conflicts.

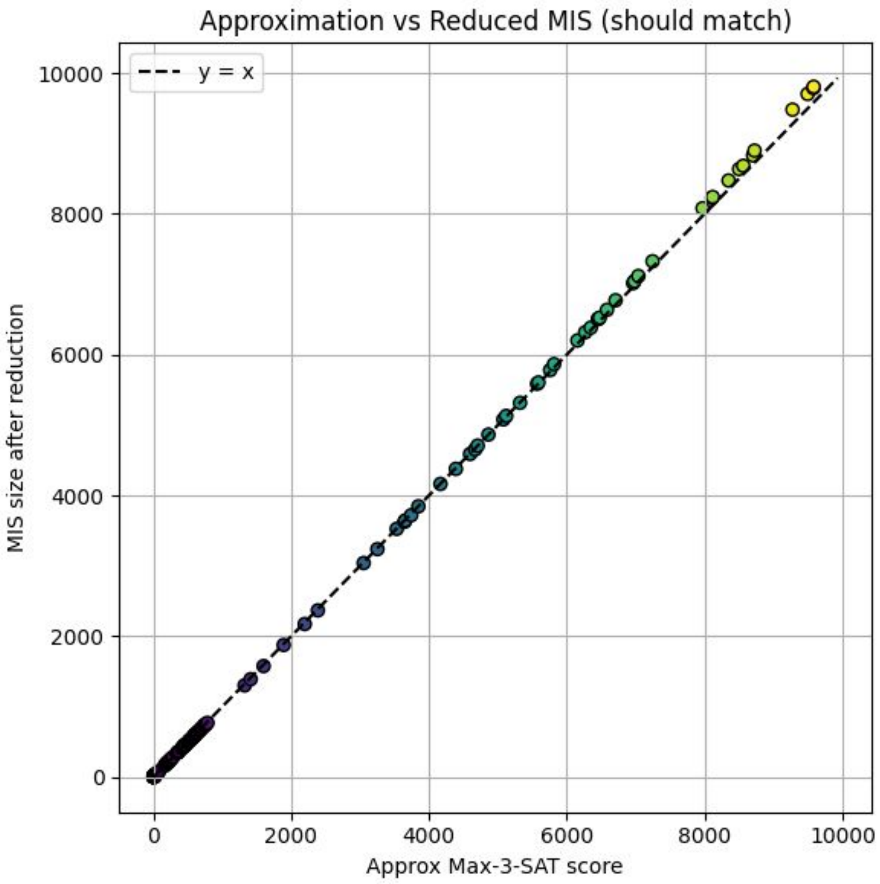


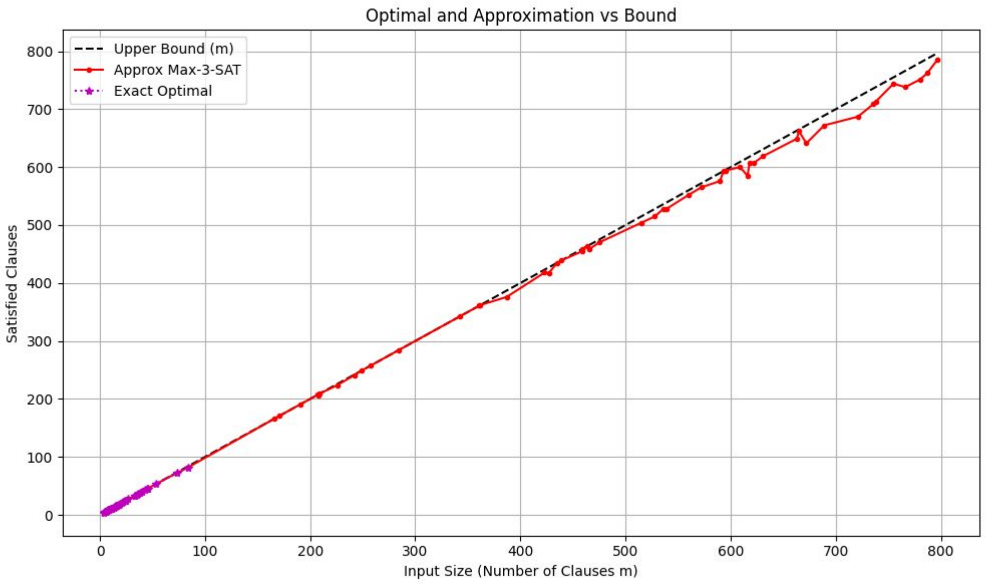
Max 3-SAT Upper Bound

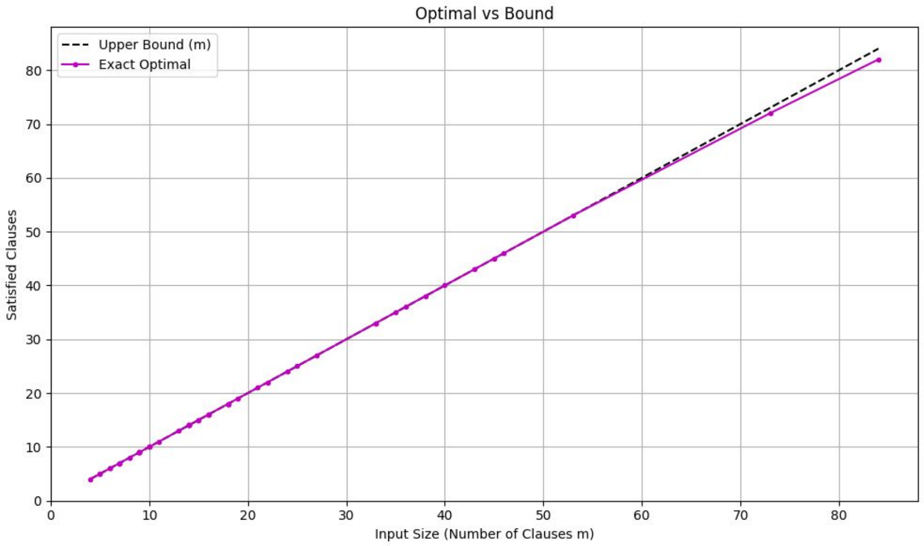
- Due to Max 3-SAT being an optimization/maximization problem we are looking to find an upper bound.
- I landed on a trivial yet effective solution for my upper bound, return the total number of clauses.
- Assuming each clause has a satisfiable literal, the maximum clauses that can be satisfied is C (all of them).
- Disadvantage: Can become looser with less ideal clauses or large quantities.



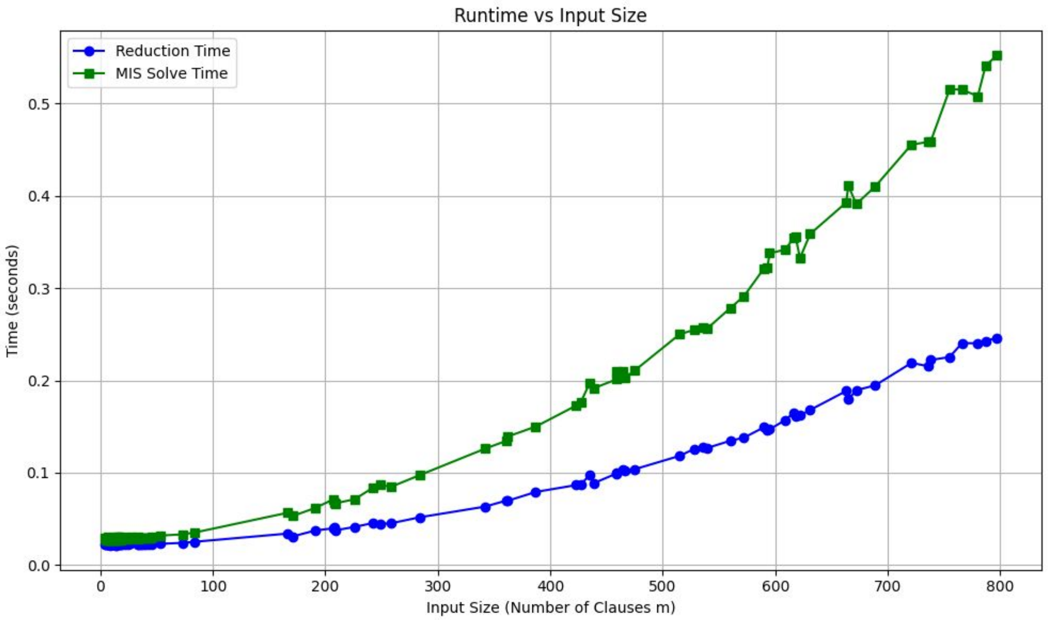
More Results







The End


Runtime vs Input Size
Reduction Time
MIS Solve Time
Time (seconds)
Input Size (Number of Clauses m)
0.0
0.1
0.2
0.3
0.4
0.5
0
100
200
300
400
500
600
700
800