Submission By - Venkatesh Duraiarasan

Contents

Exercise 4. Describe your algorithm from Exercise 3 in the form of pseudo-code

Algorithm 1: MaxSatGA Class Structure

class MaxSatGA ($clauses, num\_vars, weights, k, population\_size, pc, pm1, max\_generations, time\_budget$);

Input : Clauses, number of variables, weights, number of constraints to learn, population size, crossover probability, mutation probabilities, maximum generations, time budget, penalty, and debug flag

Output: Best solution, best fitness, and runtime

Initialize population using $initialize\_population(population\_size, num\_vars)$;

Optimize using $optimize()$;

Algorithm 2: Optimize

function optimize ($clauses, num\_vars, weights, k, population\_size, pc, pm1, max\_generations, time\_budget$);

Input : Clauses, number of variables, weights, number of constraints to learn, population size, crossover probability, mutation probabilities, maximum generations, time budget, penalty, and debug flag

Output: Best solution, best fitness, and runtime

Initialize $best\_solution$, $best\_fitness$, and $start\_time$;

for *each generation up to max_generations* **do**

  if *time elapsed exceeds time_budget* **then**

    Break;

  end

  Evaluate population fitness using $evaluate\_population(population, clauses, weights, penalty)$;

  Update $best\_solution$ and $best\_fitness$ if necessary;

  Create a new population through crossover and mutation;

  Update population;

  if *debug mode* **then**

    Print generation details;

  end

end

Return $best\_solution$, $best\_fitness$, and $runtime$;

Algorithm 3: Initialize Population

function initialize_population ($population\_size, num\_vars$);

Input : Population size and number of variables

Output: A random binary population matrix

Return a random binary matrix of size $population\_size \times num\_vars$;

Algorithm 4: Evaluate Population

function evaluate_population ($population, clauses, weights, penalty$);

Input : Population, clauses, weights, and penalty

Output: List of fitness values for the population

forall *individuals in population* **do**

  Evaluate each individual using $evaluate\_individual(individual, clauses, weights, penalty)$;

end

Return a list of fitness values;

Algorithm 5: Evaluate Individual

function evaluate_individual ($individual, clauses, weights, penalty$);
Input : Individual, clauses, weights, and penalty
Output: Fitness value of the individual
Initialize $satisfied = 0$ and $unsatisfied = 0$;
forall *clause, weight in zip(clauses, weights)* **do**
  if *clause is satisfied by individual* **then**
    Increment $satisfied$;
  end
end
Calculate fitness as $satisfied - penalty \times unsatisfied$;
Return fitness;

Algorithm 6: Crowding Selection

function crowding_selection ($fitness\_values, population\_size$);
Input : Fitness values and population size
Output: Selected individuals
Initialize an empty list *selected*;
for $i = 1$ *to* $population\_size/2$ **do**
  Randomly select two parents $p1$ and $p2$;
  Choose the parent with higher fitness and add it to *selected*;
end
Return selected individuals;

Algorithm 7: Clause Crossover

function clause_crossover ($parent1, parent2, num\_vars$);
Input : Two parent individuals and number of variables
Output: Child individual
Convert parents to strings;
Randomly select a crossover point;
Create a child by combining parts of parents;
Convert child back to a binary array;
Return child;

Algorithm 8: Mutate Hardness

function mutate_hardness ($individual, pm1, k$);
Input : Individual, mutation probability, and number of constraints
Output: Mutated individual
if *random probability* $< pm1$ **then**
    Randomly select an index idx;
    Flip the bit at idx in the individual;
end
Return individual;

Exercise 5. Experimental Analysis of Essential Parameters in a MAXSAT Algorithm

The parameters chosen for analysis were population size, crossover probability, and mutation probability, as these are crucial in genetic algorithms for MAXSAT problems.

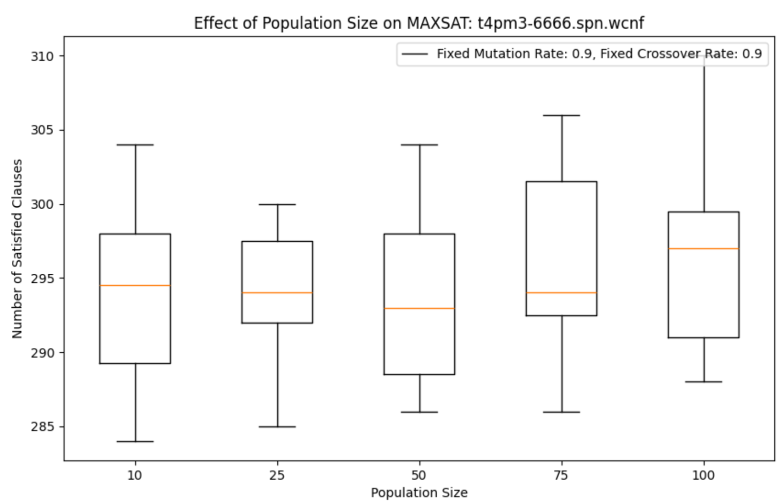
Experimental Setup **Problem Instances:** A subset of the smallest instances from the MSE17 benchmarks was selected to ensure computational feasibility.

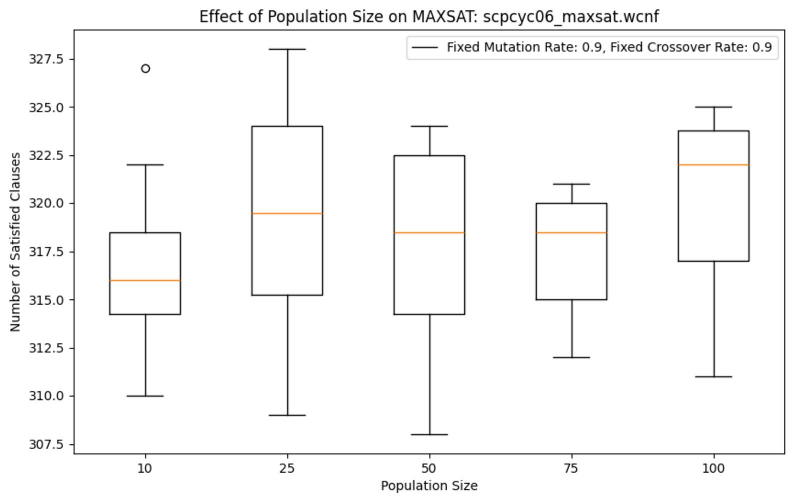
- `scpcyc06_maxsat.wcnf`
- `t4pm3-6666.spn.wcnf`

Parameter Settings:

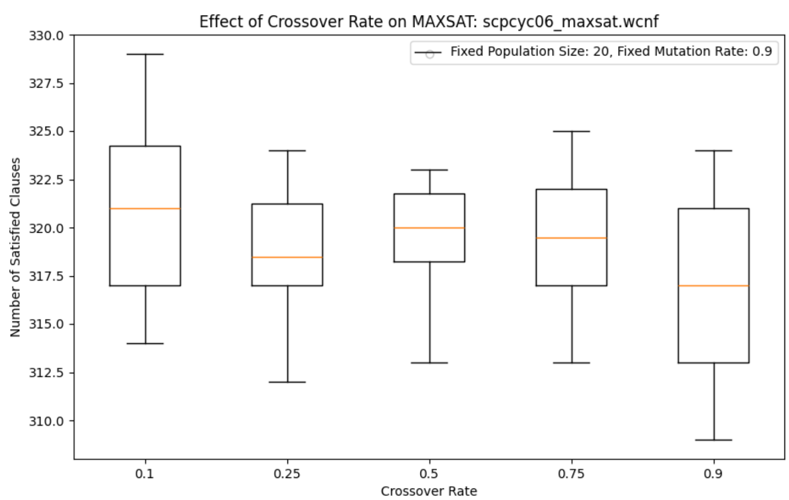
- **Population Sizes:** [10, 25, 50, 75, 100]
- **Mutation Rates**: [0.1, 0.25, 0.5, 0.75, 0.9]
- **Crossover Rates:** [0.1, 0.25, 0.5, 0.75, 0.9]
- **Timeout:** 6 seconds per repetition
- **Repetitions:** 100 runs per parameter setting and problem instance

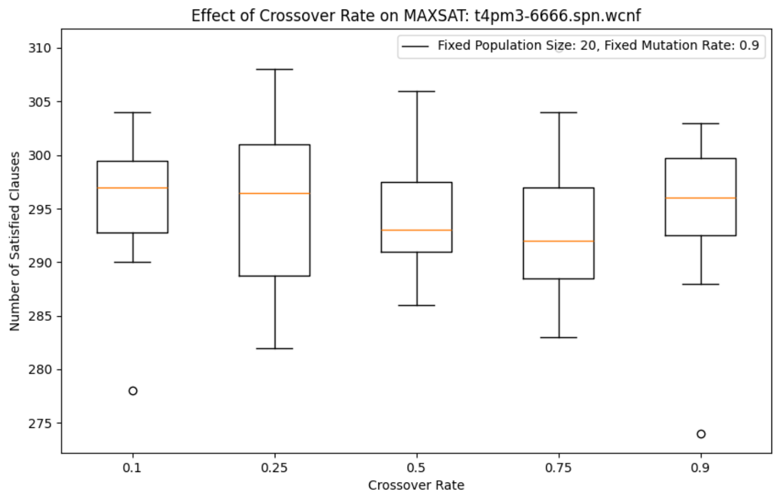
Results & Analysis **Population Size:** Larger population sizes improved solution quality by enhancing search space diversity. Increasing from 10 to 100 significantly improved results but increased computational time.



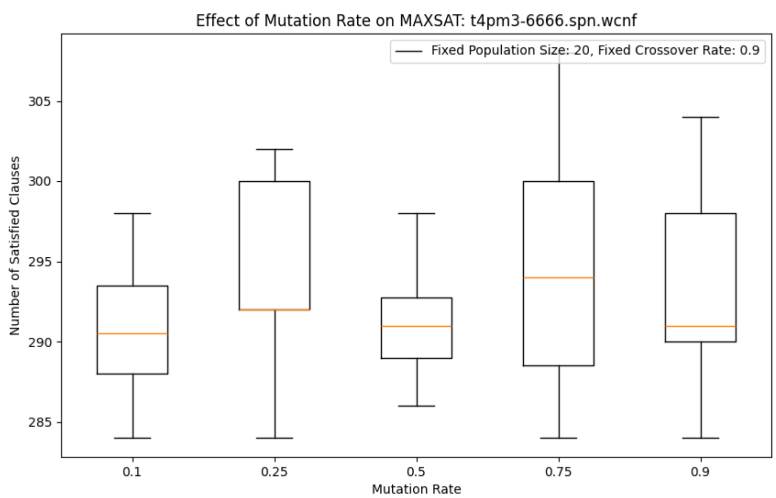


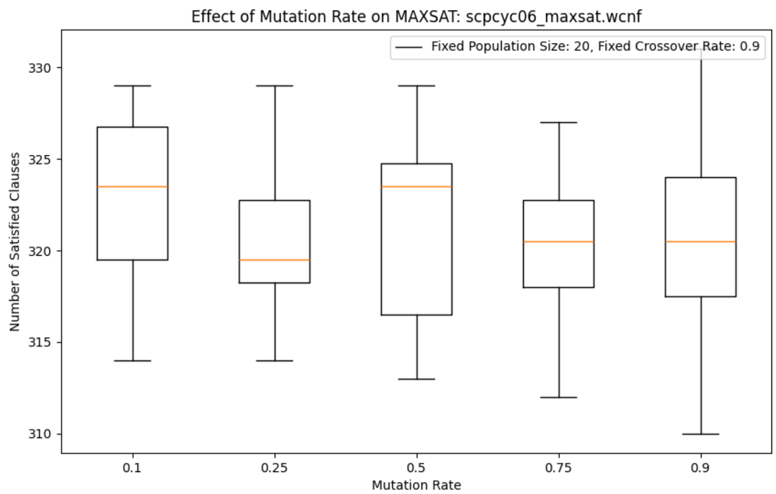
Crossover Probability: A probability of 0.1 to 0.25 balanced exploration and exploitation effectively. Higher values (e.g., 0.9) led to instability due to excessive genetic variation.





Mutation Probability: Lower rates (e.g., 0.1) limited diversity, while higher rates (e.g., 0.9) caused population instability.




Effect of Mutation Rate on MAXSAT: scpcyc06_maxsat.wcnf
Fixed Population Size: 20, Fixed Crossover Rate: 0.9
Number of Satisfied Clauses
330
325
320
315
310
0.1
0.25
0.5
0.75
0.9
Mutation Rate