Submission By - Venkatesh Duraiarasan

Contents

Task 5 : For each benchmark problem, list the average result and standard deviations obtained over 30 independent runs of each algorithm.

Table 1: Benchmark Results

Problem	Algorithm	Avg Result	Std Dev Result	Avg Time (s)
sppnw41	Simulated Annealing	11332	49.2	0.26
	BGA	15908	2446.4	2.6
	Improved BGA	11307	0.0	3.5
	Imporved BGA(stochastic ranking)	11307	0.0	47.6
sppnw42	Simulated Annealing	9041	1268.1	1.9
	BGA	13687	952.1	14.6
	Improved BGA	7666	0.0	14.0
	Imporved BGA(stochastic ranking)	7665	3.0	81.6
sppnw43	Simulated Annealing	9559	487.0	1.1
	BGA	14220	2011.1	12.5
	Improved BGA	8904	0.0	10.1
	Imporved BGA(stochastic ranking)	8904	0.0	90.3

Task 6 : Similarity and difference between the ranking replacement method and the stochastic ranking method

Difference

- Ranking Replacement focuses on the **selection** of individuals for deletion after offspring are produced, acts like a replacement algorithm.
- Stochastic Ranking focuses on the **ordering** ranking/sorting of the population, acts like a sorting algorithm. Other than unfitness and fitness value, Stochastic Ranking has p_f probabilistic value , through which ($p_f < 0.5$) we can prioritise feasible solutions (zero constraint violation) while allowing exploration of infeasible regions.

Similarity

- Both handles constraint violation.
- Both considers unfitness(constraint violation) and fitness value(objective function).

Task 7 : Compare the results from SA, the standard BGA and the improved BGA with an in-depth discussion.

Experiment details :

- trials : 30 (independent)
- Standard BGA

  - initial population size :1000
  - Iteration : 100
  - constraint violation penalty, lambda=100000
- Improved BGA and Imporved BGA with Stochastic ranking
  - initial population size :100
  - Iteration : 100
  - constraint violation penalty, lambda=100000

Problem	Algorithm	Avg Fitness	# Std Dev	Avg Time (s)	Std Dev Time (s)
Problem: sppnw41.txt					
sppnw41.txt	simulated annealing	11332	49.2	0.26	0.008
sppnw41.txt	standard bga	15908	2446.499	2.668	0.16
sppnw41.txt	improved bga	11307	0	3.487	0.666
sppnw41.txt	improved bga with stochastic ranking	11307	0	47.565	1.513
Problem: sppnw42.txt					
sppnw42.txt	simulated annealing	9041	1268.159	1.98	0.332
sppnw42.txt	standard bga	13687	952.11	14.679	2.132
sppnw42.txt	improved bga	7666	0	14.083	0.841
sppnw42.txt	improved bga with stochastic ranking	7665	3	81.627	9.45
Problem: sppnw43.txt					
sppnw43.txt	simulated annealing	9559	487.055	1.179	0.116
sppnw43.txt	standard bga	14220	2011.111	12.538	2.066
sppnw43.txt	improved bga	8904	0	10.155	0.148
sppnw43.txt	improved bga with stochastic ranking	8904	0	90.37	7.657

Figure 1: Table showing algorithms with best average fitness score (Green) and worst execution time(Red)

Overall Observations:

Fitness score analysis Improved Genetic Algorithms (ibga and ibgasr): Across all three benchmark problems (sppnw41.txt, sppnw42.txt, sppnw43.txt), the improved basic genetic algorithm (ibga) and the improved basic genetic algorithm with stochastic ranking (ibgasr) consistently **achieve the lowest average fitness values**. This indicates that the enhancements made to the basic genetic algorithm are highly effective in finding better solutions.

Stochastic Annealing (sa) is Competitive: Simulated annealing (sa) demonstrates competitive performance, particularly on sppnw41.txt, where it achieves results comparable to the improved genetic algorithms. However, its performance is less consistent across other problems.

Basic Genetic Algorithm (bga) is Outperformed: The standard basic genetic algorithm (bga) is consistently outperformed by all other algorithms in terms of solution quality (average fitness). This highlights the importance of incorporating improvements or alternative search strategies.

Impact of Stochastic Ranking (ibgasr): In terms of average fitness, the addition of stochastic ranking (ibgasr) to the improved genetic algorithm does not significantly affect the solution quality. However, it does have a noticeable impact on the execution time ie higher execution time.

Execution Time Variation There are substantial differences in execution time between the algorithms. The improved genetic algorithms (ibgasr) generally take longer than than other algorithm considered, likely due to the additional computational overhead of the stochastic ranking process.

Standard Deviation Analysis The improved genetic algorithms (ibga and ibgasr) show very low or zero standard deviations in fitness, indicating **high consistency in solution quality** across multiple runs. Simulated annealing (sa) and the basic genetic algorithm (bga) exhibit higher standard deviations, suggesting greater variability in solution quality across runs.

Initial population size Standard BGA required initial population size of 1000 for generating best feasible solution within 100 iterations. Whereas improved BGA required only initial population of 100 and converged in less than 30 iterations

Summary

- The improved genetic algorithms demonstrate superior performance in terms of solution quality and consistency.
- Simulated annealing offers a viable alternative, especially when execution time is a concern.
- Adding stochastic ranking increases runtime.

Task 4 : Introduce the SA, the standard BGA and the improved BGA.

Simulated Annealing

Pseudocode

```
=============================================================================================
FUNCTION SimulatedAnnealing
(problem, cost, neighbor, initial_solution, initial_temp, cooling_rate, iterations):
=============================================================================================

  current_solution = initial_solution
  current_cost = cost(current_solution, problem)
  best_solution = current_solution
  best_cost = current_cost
  temperature = initial_temp

  FOR i = 1 TO iterations:
    new_solution = neighbor(current_solution, problem)
    new_cost = cost(new_solution, problem)

    IF new_cost < current_cost OR Random(0, 1) < exp((current_cost - new_cost) / temperature):
      current_solution = new_solution
      current_cost = new_cost

    IF current_cost < best_cost:
      best_solution = current_solution
      best_cost = current_cost

    temperature = temperature * cooling_rate

  RETURN best_solution, best_cost
```

Standard - BGA

Pseudo code

```
=============================================================================================
FUNCTION BGA
(constraint_matrix, column_costs, iterations, population_size, penalty):
=============================================================================================
  population = InitializePopulation(population_size, num_columns)
```

```
  FOR i = 1 TO iterations:
    fitness = CalculateFitness(population, constraint_matrix, column_costs, penalty)
    population = SelectBest(population, fitness, population_size)
    parents = TournamentSelection(population, fitness)
    offspring = UniformCrossover(parents)
    offspring = Mutate(offspring)
    population = CombineAndSelect(population, offspring, fitness, population_size)
    PRINT generation, violation, min_cost, max_cost, std_cost

  best_solution = population[0]
  best_cost = fitness[0]
  best_violation = violation[0]

  RETURN best_solution, best_cost, best_violation
```

Improved - BGA

Pseudo code

```
================================================================================================
FUNCTION Improved BGA
(constraint_matrix, column_costs, iterations, population_size, penalty):
================================================================================================

  population = InitializePopulation(population_size, problem_data) #pseudo Random intializsation
  best_solution = NULL
  best_fitness = INFINITY
  best_violation = INFINITY

  FOR i = 1 TO iterations:
    fitness, violation = EvaluatePopulation(population, constraint_matrix, column_costs, penalty)

    FOR j = 1 TO population_size:
      IF violation[j] == 0 AND (best_violation > 0 OR fitness[j] < best_fitness):
        best_fitness = fitness[j]
        best_solution = population[j]
        best_violation = violation[j]
      ELSE IF best_violation > 0 AND violation[j] < best_violation:
        best_violation = violation[j]
        best_solution = population[j]
        best_fitness = fitness[j]

    parents = TournamentSelection(population, fitness)
    offspring = UniformCrossover(parents)
    offspring = Mutate(offspring)

    new_offspring = []
    FOR each child in offspring:
      improved_child = HeuristicImprovement(child, problem_data)
      IF improved_child is not a duplicate in population:
        new_offspring.append(improved_child)

    FOR each child in new_offspring:
      child_fitness, child_violation = EvaluateIndividual(child, constraint_matrix, column_costs, pe
      replace_index = RankingReplacement(population, fitness, violation, child_fitness, child_violat
      population[replace_index] = child
      fitness[replace_index] = child_fitness
```

```
    violation[replace_index] = child_violation

    IF (child_violation == best_violation == 0 AND child_fitness < best_fitness) OR (best_violatio
      best_fitness = child_fitness
      best_solution = child
      best_violation = child_violation

  PRINT generation, best_violation, best_fitness, max(fitness), std(fitness)

RETURN best_solution, best_fitness, best_violation
```

Flowcharts Continued below...

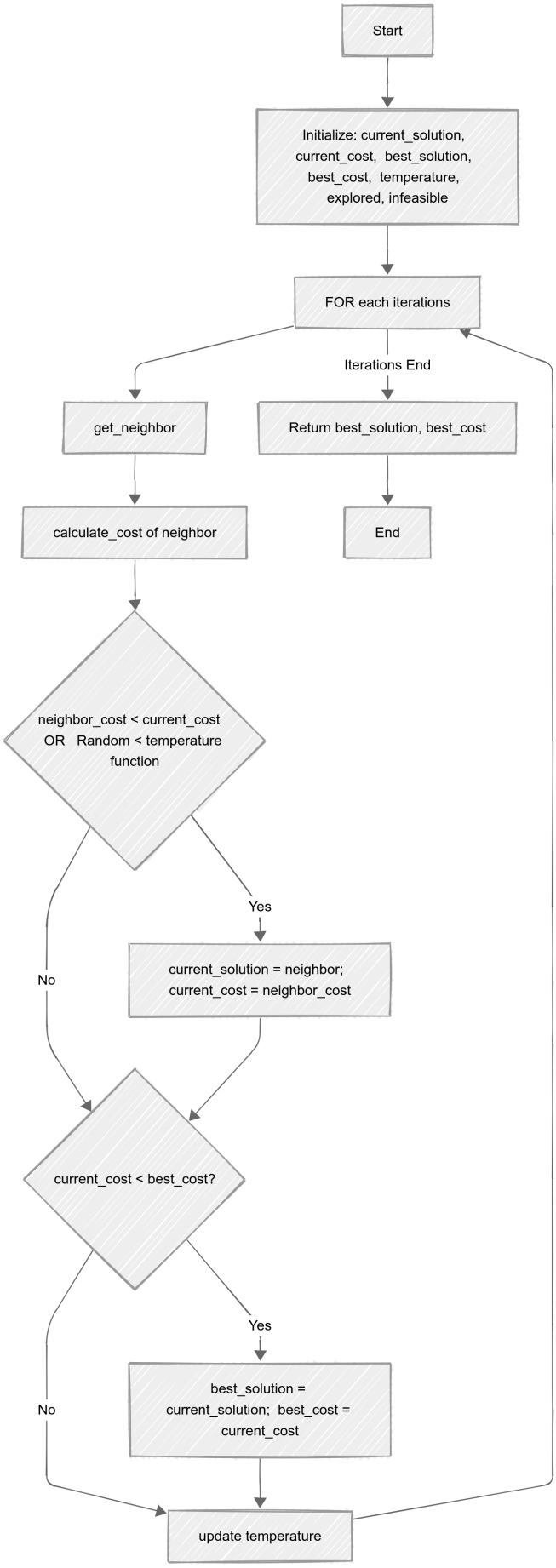

Figure 2: Simulated Annealing - flow chart

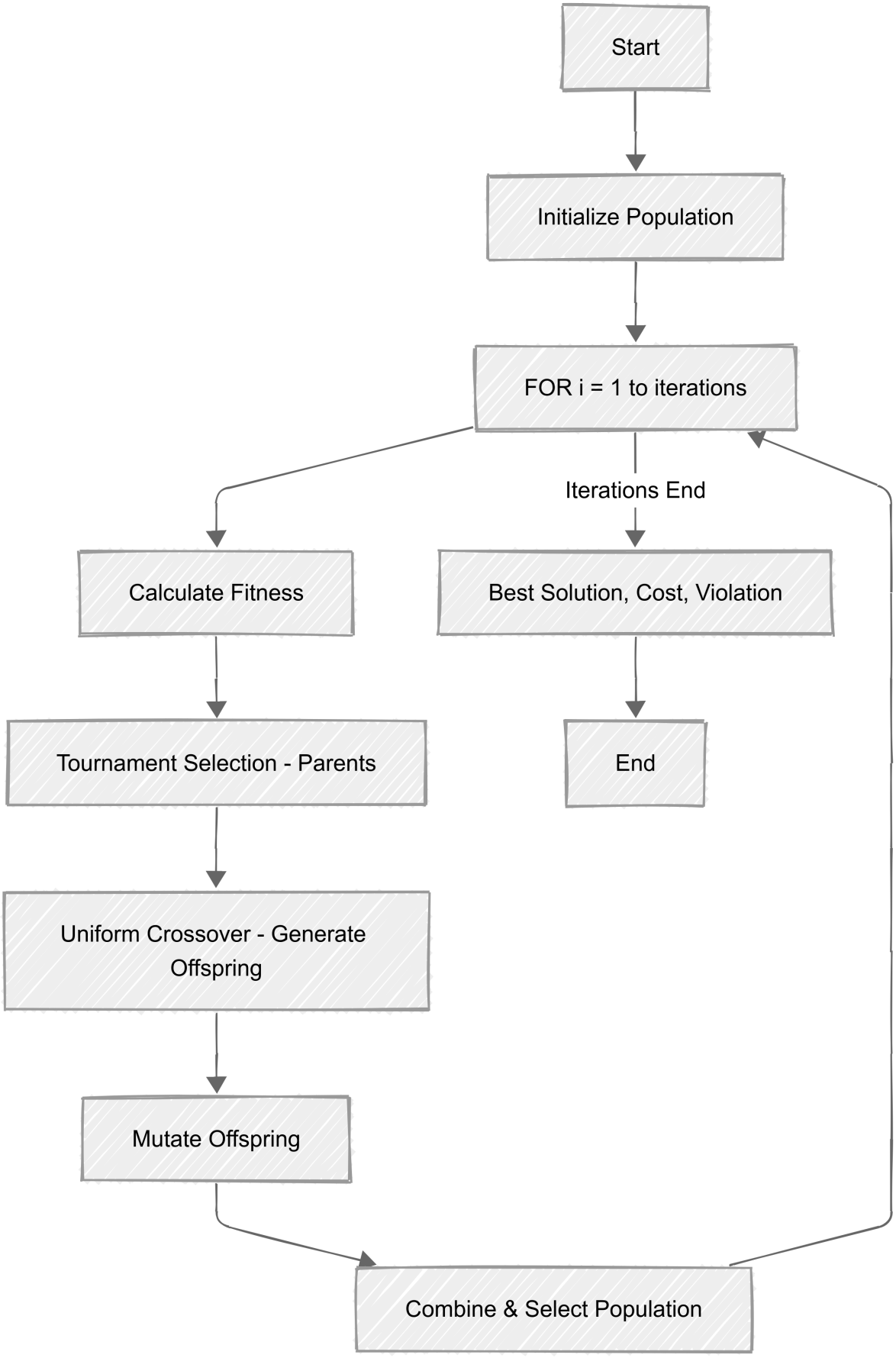

Figure 3: Standard BGA - flow chart

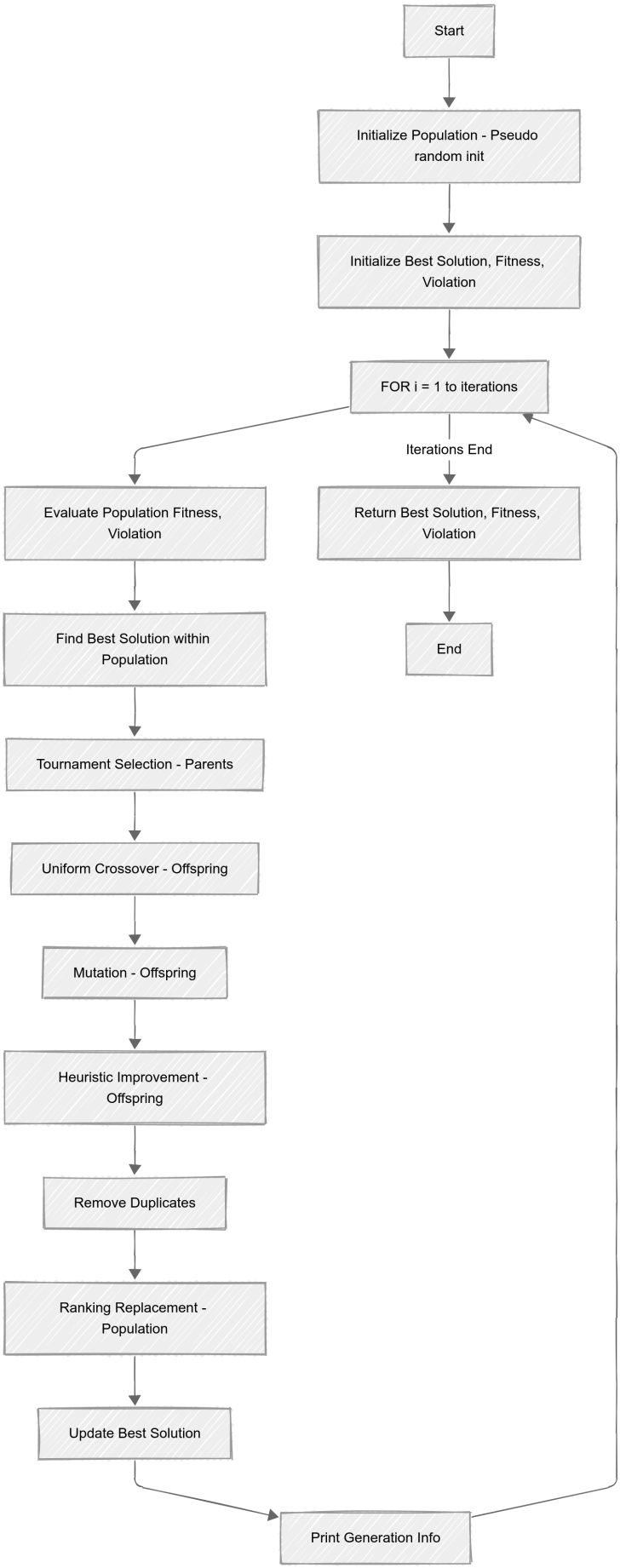

Figure 4: Improved BGA- flow chart