Computational Intelligence
CS 451

Assignment 01

Dr. Syeda Saleha Raza

Muhammad Ibad - mi08440
Nehal Naeem Haji - nh07884

Contents

0.1 Objective . . . 3

1 Travelling Salesman Problem **3**
1.1 Problem . . . 3
1.2 Chromosome Representation . . . 3
1.3 Fitness Function . . . 3
1.4 Results . . . 4
1.4.1 Parameters . . . 4
1.4.2 Plots . . . 4
1.5 Analysis . . . 8
1.6 Optimal Solution . . . 9

2 Job-Shop Scheduling **10**
2.1 Problem . . . 10
2.2 Chromosome Representation . . . 10
2.3 Fitness Function . . . 11
2.4 Results . . . 12
2.4.1 Parameters . . . 12
2.4.2 Plots . . . 12
2.5 Analysis . . . 24
2.5.1 Random Selection . . . 24
2.5.2 Parent and Survivor Selection Schemes . . . 24
2.5.3 Variability in Average Fitness . . . 25
2.5.4 Different Configurations of Parameters . . . 25
2.5.5 Conclusion . . . 25
2.6 Optimal Solution . . . 25

3 Mona Lisa problem **27**
3.1 Chromosome representation . . . 27
3.2 Fitness function . . . 27
3.3 Images . . . 28
3.4 Analysis and Findings . . . 38
3.5 Challenges and Improvements . . . 39

4 Conclusion **39**

0.1 Objective

The purpose of this assignment is to provide students an insight of global optimization using Evolutionary Algorithms (EA). This exercise will enable them to address some popular computing problems that can be mapped to several real-world problems and are known to be computationally hard. The process will expose them to the overall process of EA, its underlying challenges and the impact of different parameters and selection schemes.

1 Travelling Salesman Problem

1.1 Problem

The Travelling Salesman Problem is an NP-Hard problem in combinatorial optimization which states:
"Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?"

Our objective is to apply an evolutionary algorithm to find the shortest possible route, minimizing the total distance.

1.2 Chromosome Representation

The chromosome representation for this problem is defined as a sequence of numbers, where each number corresponds to a specific city. The order of the numbers represents a complete cycle through all the cities. For example:

$$[54, 34, 2, \ldots, 54]$$

1.3 Fitness Function

The fitness function for TSP evaluates the total distance traveled of all the candidate solutions in the population. It uses two helper function to calculate this.

DistanceFormula calculates the Euclidean distance between two cities given their coordinates.

TotalDistance computes the total distance of a route (chromosome) by summing the distances between consecutive cities. It also adds the distance from the last city back to the first city to complete the cycle. The returned value is the negative of the total distance to convert it into a maximization problem.

FitnessFunction applies the *TotalDistance* function to each chromosome in the population, returning a list of fitness values. The fitness of each chromosome is essentially the negative total distance.

Following are the pseudo codes of the three functions.

function DistanceFormula(city1, city2)
  $(x_1, y_1) \leftarrow$ coordinates of city1
  $(x_2, y_2) \leftarrow$ coordinates of city2
  return $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
end function

function TotalDistance(chromosome)
  $total\_distance \leftarrow 0$
  for $i \leftarrow 0$ to length of chromosome $- 1$ **do**
    $total\_distance \leftarrow total\_distance +$ DistanceFormula$(chromosome[i], chromosome[i+1])$
  end for
  Add distance from last city back to the first city
  return $-total\_distance$ ▷ Negative for maximization
end function

function FitnessFunction
  return list of total distances for all chromosomes in the population
end function

1.4 Results

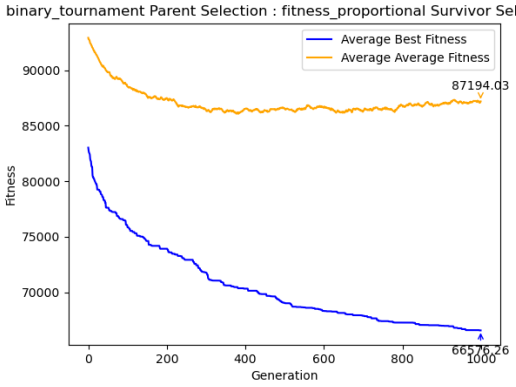
1.4.1 Parameters

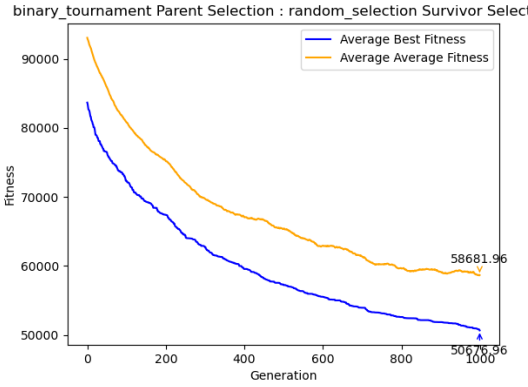
Following are the parameters used for all the plots of various combinations of selection schemes.

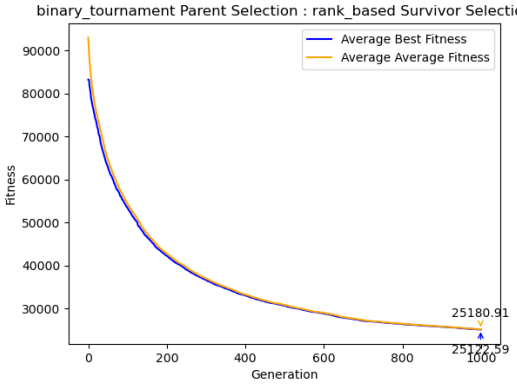
```
population_size = 500
num_generations = 1000
mutation_rate = 0.5
num_offspring = 70
num_iterations = 10
```

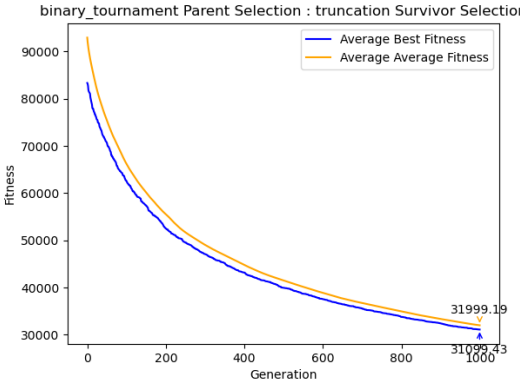
1.4.2 Plots

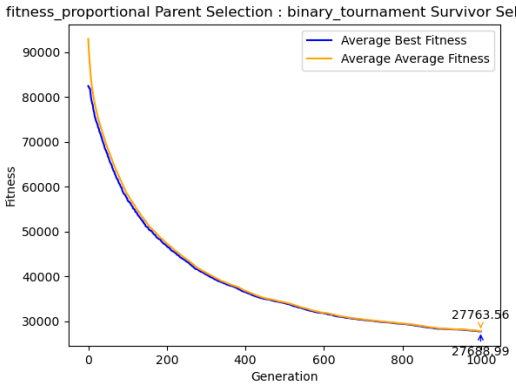
Plots of Average Best Fitness so Far and Average Fitness so Far for various combinations of schemes

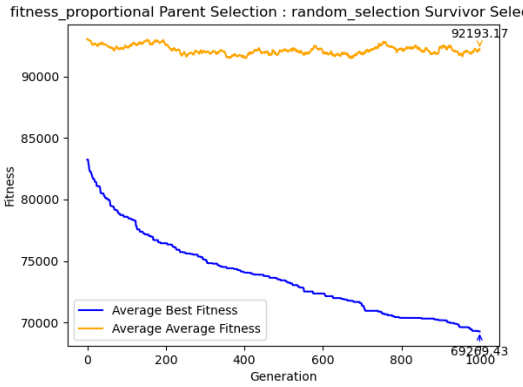

binary_tournament Parent Selection : fitness_proportional Survivor Selectio
Average Best Fitness
Average Average Fitness
87194.03
66576.26
Fitness
Generation

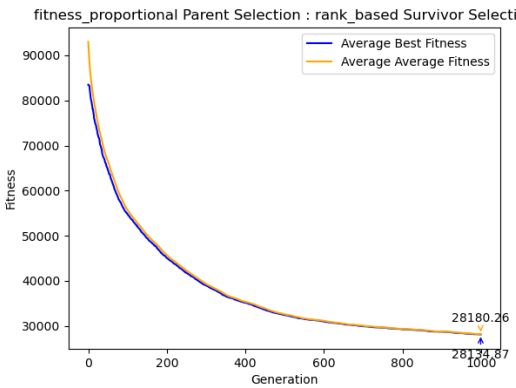

binary_tournament Parent Selection : random_selection Survivor Selection
Average Best Fitness
Average Average Fitness
58681.96
50676.96
Fitness
Generation

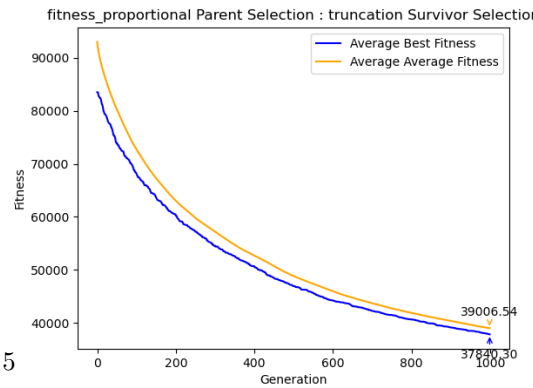

binary_tournament Parent Selection : rank_based Survivor Selection
Average Best Fitness
Average Average Fitness
25180.91
25122.59
Fitness
Generation

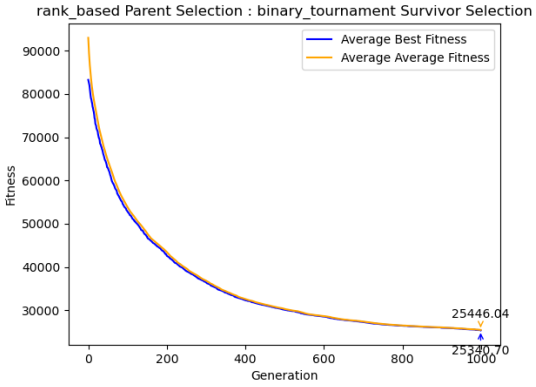

binary_tournament Parent Selection : truncation Survivor Selection
Average Best Fitness
Average Average Fitness
31999.19
31099.43
Fitness
Generation

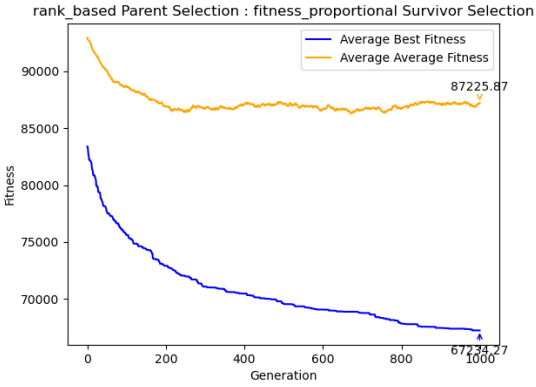

fitness_proportional Parent Selection : binary_tournament Survivor Selectio
Average Best Fitness
Average Average Fitness
27763.56
27688.99
Fitness
Generation

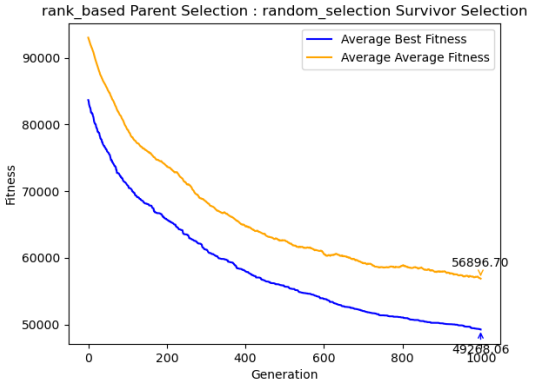

fitness_proportional Parent Selection : random_selection Survivor Selection
92193.17
Average Best Fitness
Average Average Fitness
69289.43
Fitness
Generation

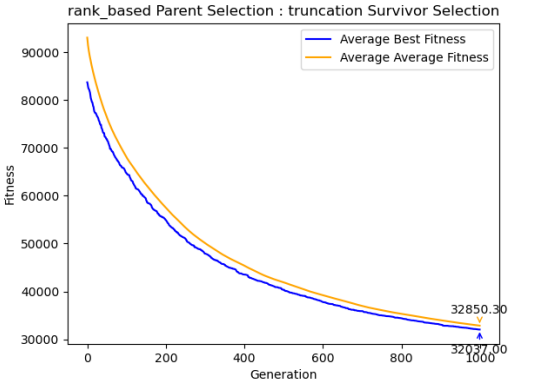

fitness_proportional Parent Selection : rank_based Survivor Selection
Average Best Fitness
Average Average Fitness
28180.26
28134.87
Fitness
Generation

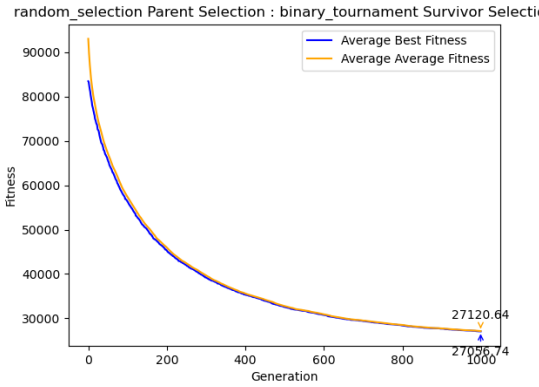

fitness_proportional Parent Selection : truncation Survivor Selection
Average Best Fitness
Average Average Fitness
39006.54
37840.30
Fitness
Generation

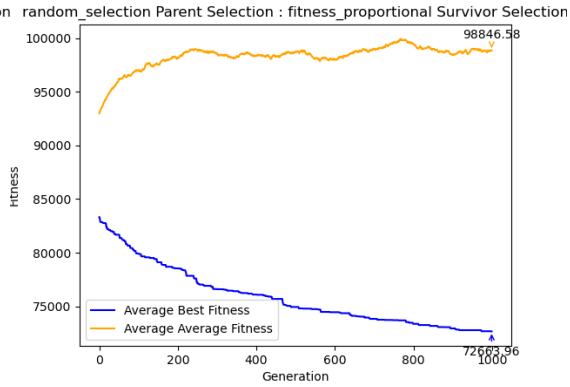

rank_based Parent Selection : binary_tournament Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
25446.04
25340.70

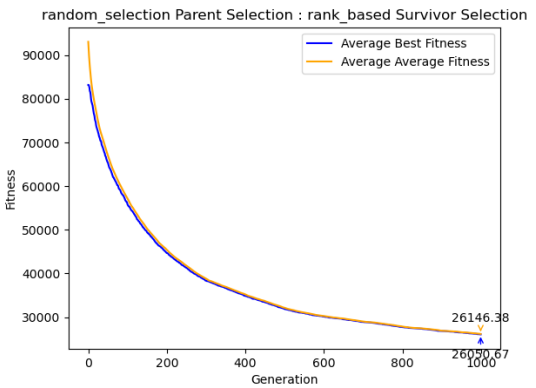

rank_based Parent Selection : fitness_proportional Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
87225.87
67234.27

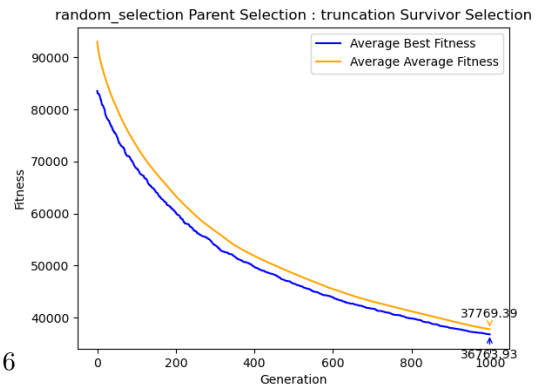

rank_based Parent Selection : random_selection Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
56896.70
49288.06

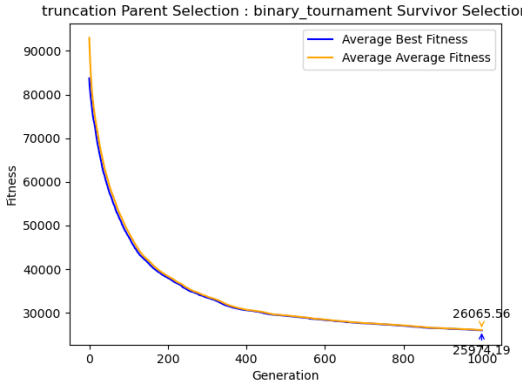

rank_based Parent Selection : truncation Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
32850.30
32037.00

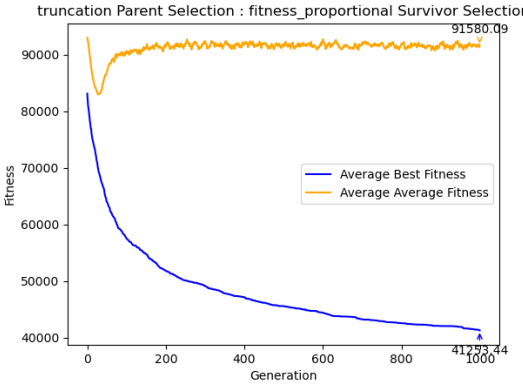

random_selection Parent Selection : binary_tournament Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
27120.64
27056.74

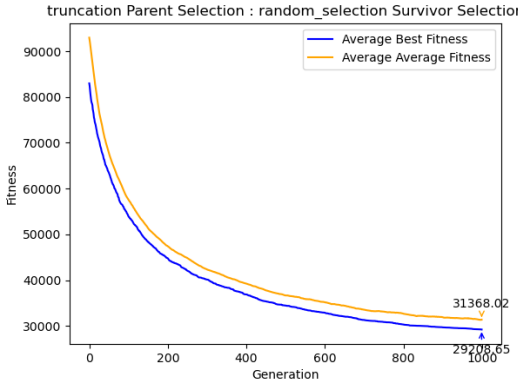

random_selection Parent Selection : fitness_proportional Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
98846.58
72665.96

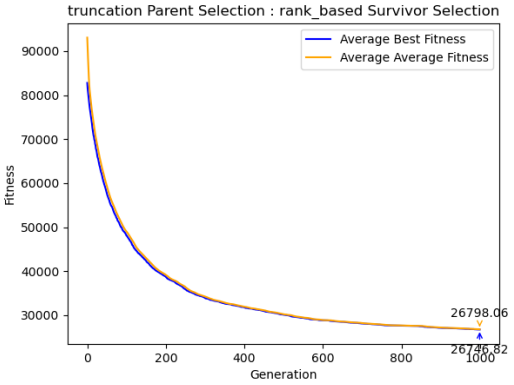

random_selection Parent Selection : rank_based Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
26146.38
26050.67


random_selection Parent Selection : truncation Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
37769.39
36783.93


truncation Parent Selection : binary_tournament Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
26065.56
25974.19


truncation Parent Selection : fitness_proportional Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
91580.09
41293.44


truncation Parent Selection : random_selection Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
31368.02
29208.65


truncation Parent Selection : rank_based Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
26798.06
26736.82

Parent	Survivor	Best Fitness	Average Fitness
Binary	Binary	26787	26830
Binary	Fitness Prop	66576	87194
Binary	Random	50676	58681
Binary	**Rank Based**	**25122**	**25180**
Binary	Truncation	31099	31999
Fitness Prop	Binary	27688	27763
Fitness Prop	**Fitness Prop**	**74783**	**100106**
Fitness Prop	Random	69269	92193
Fitness Prop	Rank Based	28134	28180
Fitness Prop	Truncation	37840	39006
Random	Binary	27056	27120
Random	Fitness Prop	72663	98846
Random	Random	66477	90151
Random	Rank Based	26050	26146
Random	Truncation	36763	37769
Rank Based	Binary	25340	25446
Rank Based	Fitness Prop	67234	87225
Rank Based	Random	49268	56896
Rank Based	Rank Based	26342	26383
Rank Based	Truncation	32037	32850
Truncation	Binary	25974	26065
Truncation	Fitness Prop	41253	91580
Truncation	Random	29208	31368
Truncation	Rank Based	26746	26798
Truncation	Truncation	25844	26258

Table 1: Comparison of Parent and Survivor Selection Methods

1.5 Analysis

Fitness Proportional Selection and Random Selection:
This combination has resulted in relatively high best and average fitness values. The graph especially suggests that using random selection for survivors, regardless of the parent selection scheme, might not be effective for this problem. However, it still performs better than several other combinations, likely due to its balance between exploration and exploitation.

Truncation and Truncation:
This combination heavily emphasized exploitation by consistently selecting only the best parents and offspring. In our case, this approach proved effective, making it one of the best-performing combinations. However, it is still far from the optimal solution, indicating the need to incorporate an explorative element into the selection schemes to achieve better outcomes.

Random and Random, Random and FPS:
These combinations yielded high best and average fitness values, indicating poor optimization performance. This suggests that Random and FPS are ineffective survivor selection schemes for this problem, likely because they lack a structured approach to retaining high-quality candidate solutions. The randomness in these schemes may hinder the algorithm's ability to maintain and

build upon promising solutions.

Random and RBS, Random and BT, Random and Truncation:
These combinations converged to sub-optimal solutions, suggesting that even with random parent selection, the optimization process can still function as long as the next generation includes good candidate solutions. However, the effectiveness is reduced due to the randomness in parent selection. These graphs also indicate that, among the five selection schemes, these three for survivor selection are among the most effective.

BT - BT, BT - RB, BT - T, FPS - BT, FPS - RB, RB - B, RB - RB, RB - T, T - BT, T - R, T - RB, T - T:
These combination yielded relatively low best and average fitness values, which could be further optimized through additional runs and experimentation. These selections strikes a good balance between exploration and exploitation which contribute highly in a smooth convergence to optimal and sub-optimal solutions.

FPS and FPS:
Lastly, this combination produced the lowest best and average fitness values, highlighting the ineffectiveness of FPS for this problem as both a parent and survivor selection scheme. Its poor performance can be attributed to the inherent randomness in the selection process, which may lead to the retention of sub-optimal solutions and hinder convergence. This emphasizes the importance of balancing exploration and exploitation in selection strategies.

1.6 Optimal Solution

After optimizing using all the combinations of selection schemes, we chose Rank-based Selection as our Parent Scheme and Truncation Selection as our Survivor Scheme. To further enhance the optimization process, we allowed each iteration to inherit the population from the previous iteration's generation. This ensured that every new iteration built upon the improvements of the last. For further optimization, we implemented a strategy where the random population was run for a large number of iterations and generations. Afterward, the best solutions from these runs were selected and combined with a new random population, and the process was repeated. Essentially, this approach involved running the optimization overnight to ensure the population continuously evolved.

Despite the persistent improvements, the optimization ultimately converged around the 10k mark, indicating that while the strategy maintained steady progress, it reached a limit under the current setup.

Parameters Used for Best Optimization:

```
Population size: 200
Number of generations: 100
Mutation rate: 0.5
Number of offspring: 70
Number of iterations: 100
Parent selection scheme: Rank-based selection
Survival selection scheme: Truncation selection
```

Best Fitness: **10930**

Best Chromosome: [87, 102, 91, 103, 106, 105, 107, 108, 116, 117, 121, 120, 123, 124, 128, 143, 155, 151, 158, 162, 171, 185, 193, 192, 191, 189, 190, 187, 184, 177, 181, 188, 183, 186, 194, 182, 176, 172, 169, 163, 161, 156, 145, 140, 134, 132, 130, 111, 99, 94, 90, 98, 86, 85, 65, 20, 63, 36, 59, 62, 71, 76, 75, 78, 72, 74, 69, 26, 24, 33, 28, 60, 57, 45, 37, 40, 43, 47, 58, 56, 53, 48, 38, 41, 46, 52, 54, 55, 49, 50, 42, 44, 35, 31, 32, 30, 19, 15, 12, 10, 9, 5, 3, 2, 4, 1, 6, 8, 16, 13, 23, 25, 14, 11, 7, 17, 21, 18, 22, 29, 27, 34, 39, 51, 61, 67, 73, 66, 68, 64, 70, 77, 84, 81, 79, 83, 88, 93, 96, 95, 92, 97, 100, 110, 112, 115, 118, 129, 133, 135, 131, 136, 148, 160, 166, 170, 180, 178, 175, 173, 174, 179, 164, 149, 146, 142, 137, 127, 114, 109, 113, 119, 122, 141, 144, 154, 157, 165, 168, 167, 159, 147, 152, 153, 150, 139, 138, 126, 125, 104, 101, 89, 82, 80]

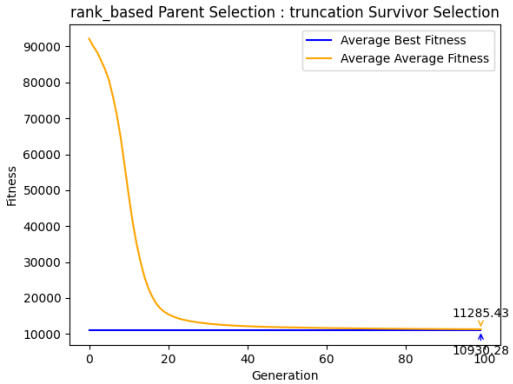


Figure 4: Optimal Solution

2 Job-Shop Scheduling

2.1 Problem

The job-shop scheduling problem is an optimization problem and is a variant of optimal job scheduling. In a general job scheduling problem, we are given n jobs of varying processing times, which need to be scheduled on m machines with varying processing power, while trying to minimize the makespan - the total length of the schedule. This variant also contains a set of *operations* which need to be processed in a specific order.

Our objective is to apply an evolutionary algorithm to minimize the makespan, the total time required to complete all jobs.

2.2 Chromosome Representation

The chromosome representation for this problem is defined as a sequence of jobs represented by a unique job ID. The order of the numbers represents the order in which operations from different jobs should be executed. It indicates which job's operation gets scheduled next.

The chromosome's length is the product of the number of machines and the number of jobs as we are scheduling a set of jobs across multiple machines, and each operation for each job needs to be scheduled.

The sequence can be shuffled and mutated to explore various possible schedules as long as the permutations follow the constraint that each job's operation needs to be executed in the correct order.

2.3 Fitness Function

The primary goal of the fitness function is to calculate the makespan for the whole population. The makespan is a crucial metric for evaluating the efficiency of schedules.

For each chromosome in the population, the fitness function simulates the execution of jobs based on the order specified. It tracks the completion times for each job and the availability times of each machine. The makespan for a chromosome is the maximum completion time across all jobs. Furthermore, to align with the minimization goal in the evolutionary algorithm, the fitness function returns negative makespan values for the entire population. This allows the evolutionary algorithm to treat the problem as maximizing fitness.

Below is the pseudo-code of the fitness function:

Algorithm 1 Fitness Function for Job Shop Scheduling

Input: Population of chromosomes
Output: List of fitness values (negative makespan) for the population

$makespans \leftarrow []$ ▷ Initialize list to store makespans
for each chromosome in population **do**

  $job\_completion\_time \leftarrow [0, 0, \ldots, 0]$ ▷ Initialize job completion times
  $machine\_available\_time \leftarrow [0, 0, \ldots, 0]$ ▷ Initialize machine availability times
  $job\_operation\_index \leftarrow [0, 0, \ldots, 0]$ ▷ Initialize operation index for each job

  for each gene in chromosome **do**
    $job\_id \leftarrow gene$
    $operation\_index \leftarrow job\_operation\_index[job\_id]$
    $machine\_id, processing\_time \leftarrow data[job\_id][operation\_index]$

    $earliest\_start\_time \leftarrow \max(job\_completion\_time[job\_id], machine\_available\_time[machine\_id])$
    $completion\_time \leftarrow earliest\_start\_time + processing\_time$

    $job\_completion\_time[job\_id] \leftarrow completion\_time$
    $machine\_available\_time[machine\_id] \leftarrow completion\_time$
    $job\_operation\_index[job\_id] \leftarrow job\_operation\_index[job\_id] + 1$
  end for
  $makespan \leftarrow \max(job\_completion\_time)$
  $makespans.append(-makespan)$ ▷ Store negative makespan to minimize it
end for
return makespans ▷ Return list of fitness values

2.4 Results

2.4.1 Parameters

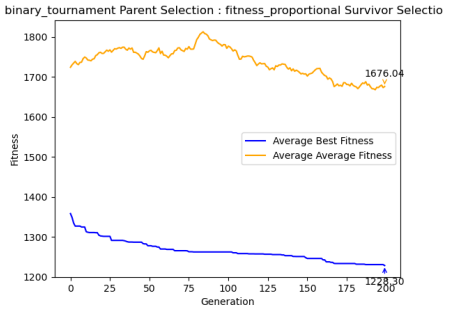
Following are the parameters used for all the plots of various combinations of selection schemes for all instances.

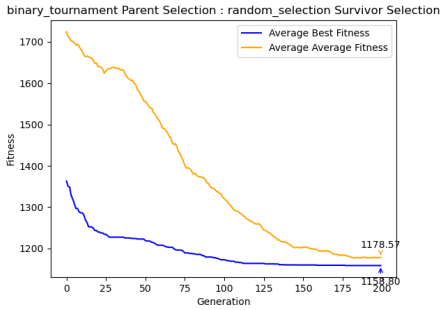
```
population_size = 200
num_generations = 200
mutation_rate = 0.5
num_offspring = 70
num_iterations = 10
```

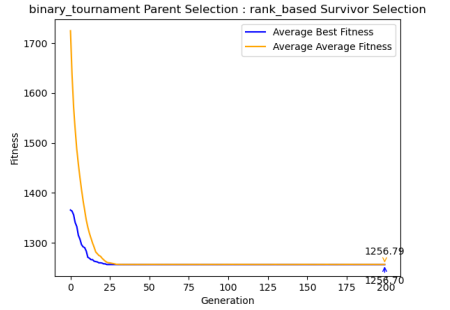
2.4.2 Plots

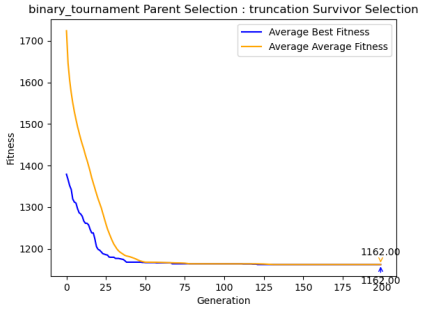
Plots of Average Best Fitness so Far and Average Fitness so Far for various combinations of schemes

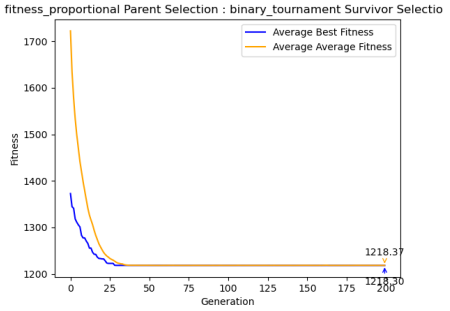
Instance 1 (ft10)

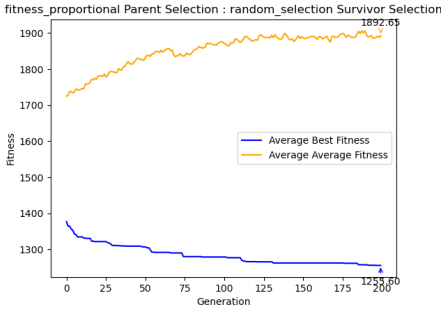


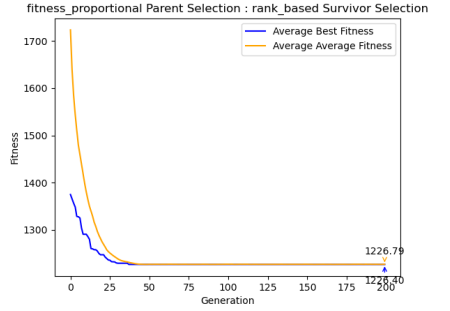


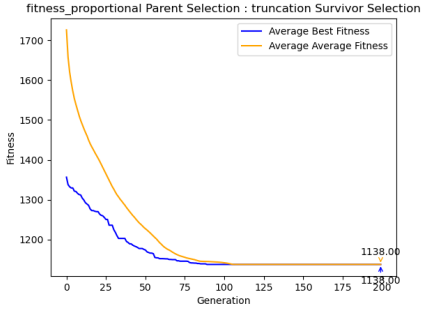


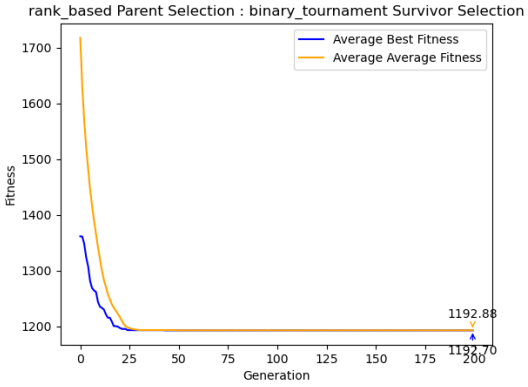


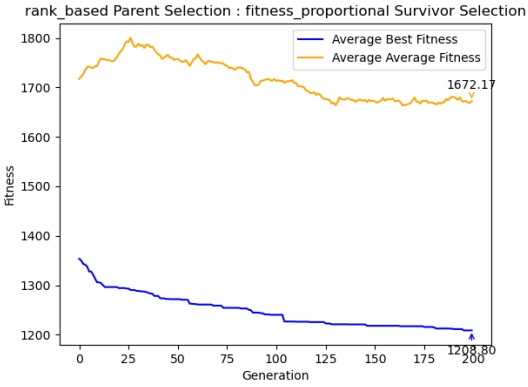


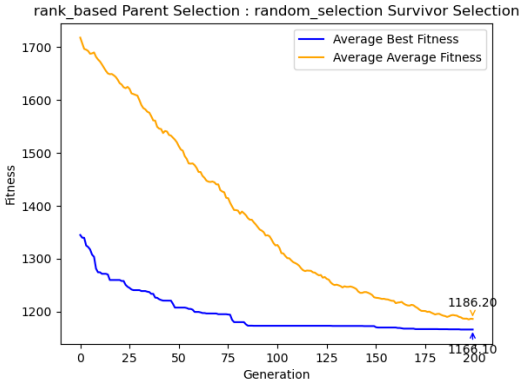


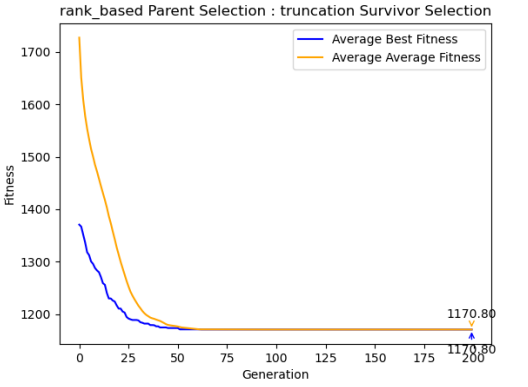


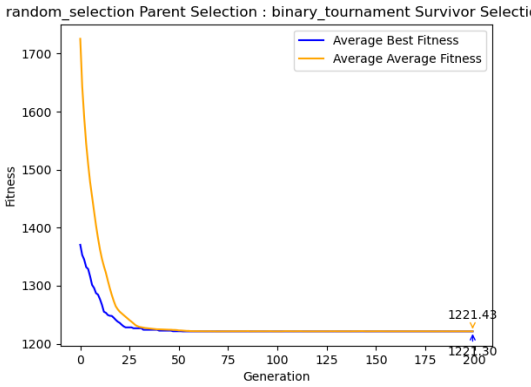


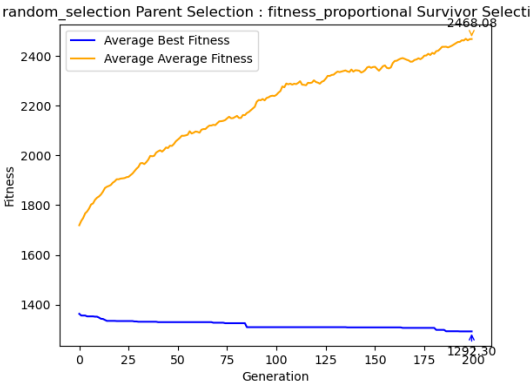

rank_based Parent Selection : binary_tournament Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1192.88
1192.70

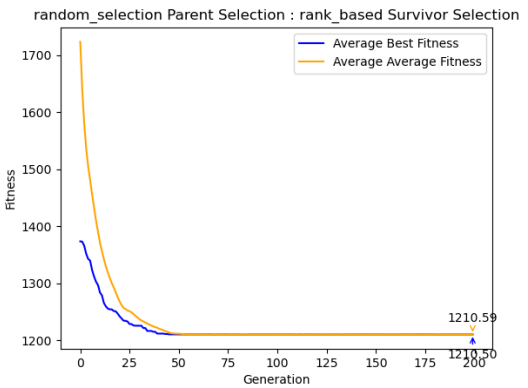

rank_based Parent Selection : fitness_proportional Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1672.17
1208.80

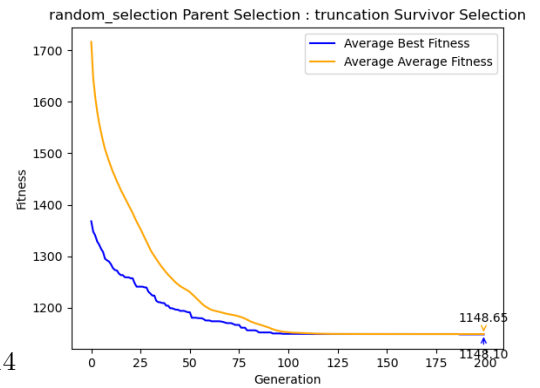

rank_based Parent Selection : random_selection Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1186.20
1168.10

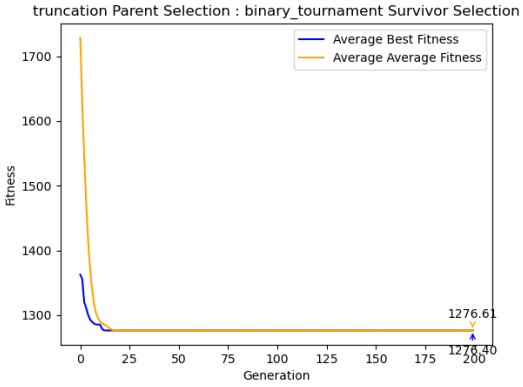

rank_based Parent Selection : truncation Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1170.80
1170.80

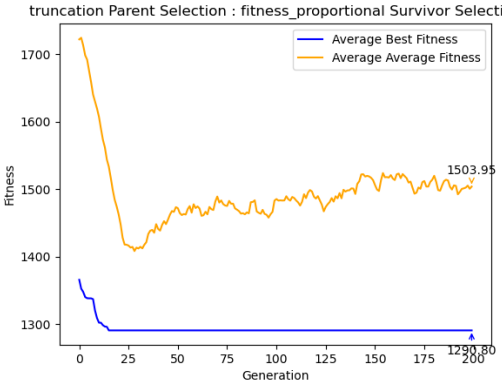

random_selection Parent Selection : binary_tournament Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1221.43
1221.30

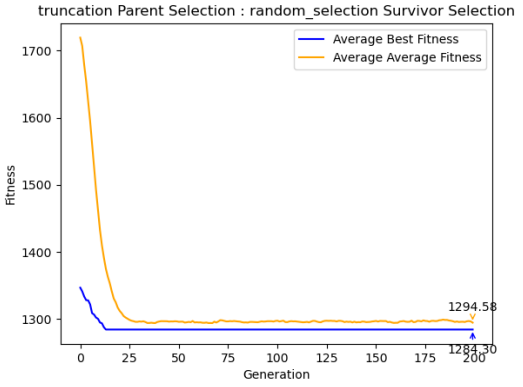

random_selection Parent Selection : fitness_proportional Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
2468.08
1292.30

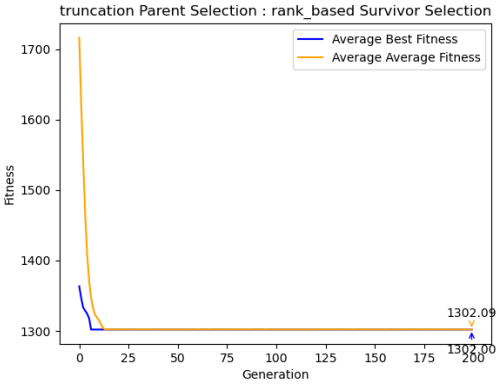

random_selection Parent Selection : rank_based Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1210.59
1210.50


random_selection Parent Selection : truncation Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1148.65
1148.10


truncation Parent Selection : binary_tournament Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1276.61
1276.40


truncation Parent Selection : fitness_proportional Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1503.95
1290.80

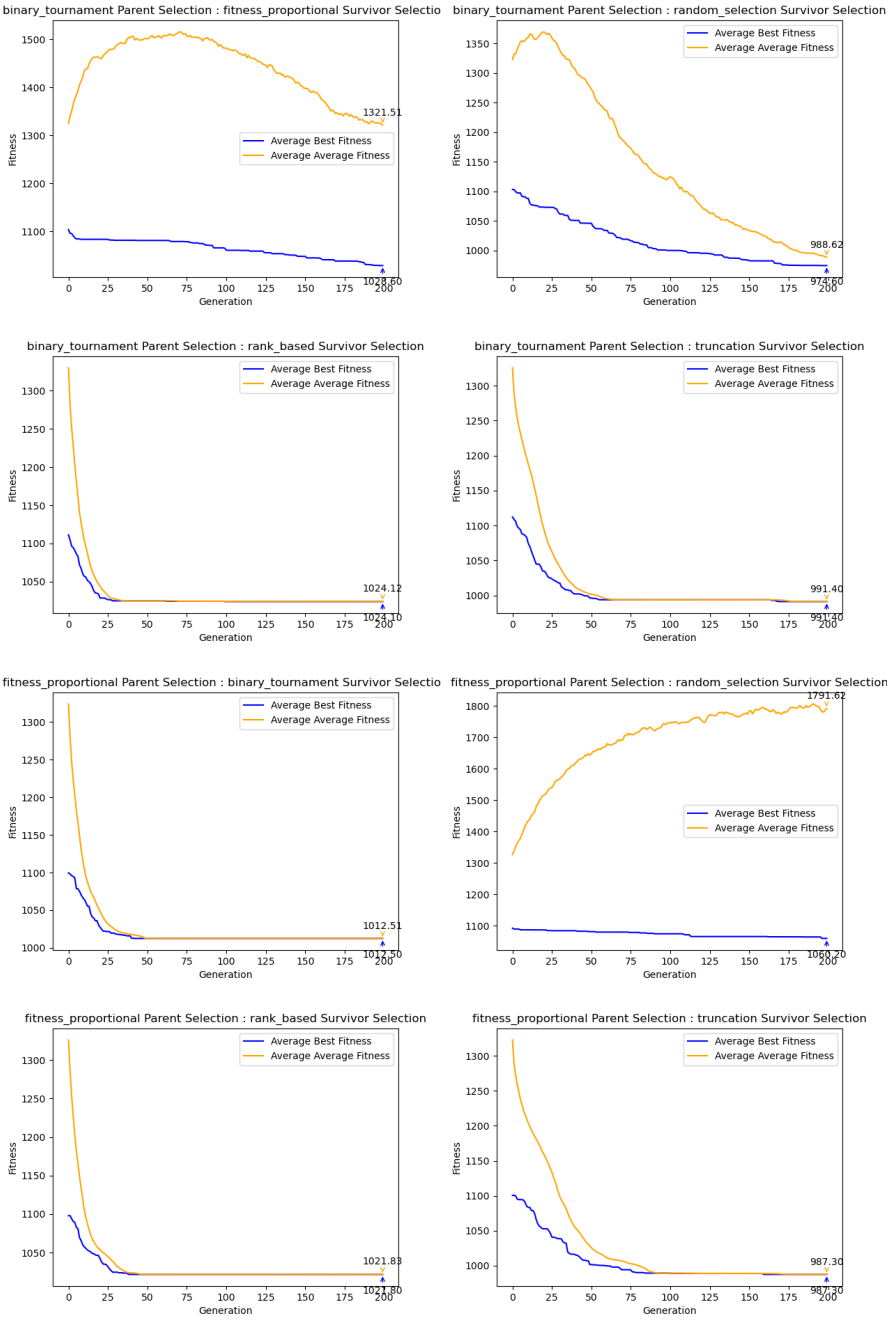

truncation Parent Selection : random_selection Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1294.58
1284.30

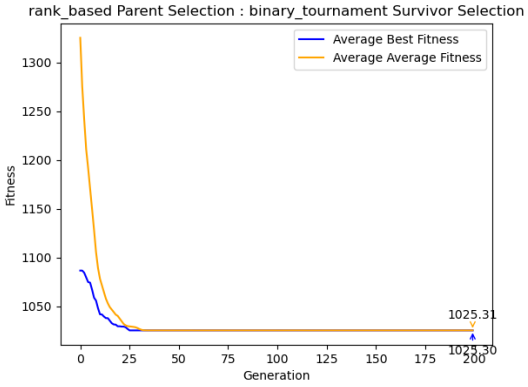

truncation Parent Selection : rank_based Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1302.09
1302.00

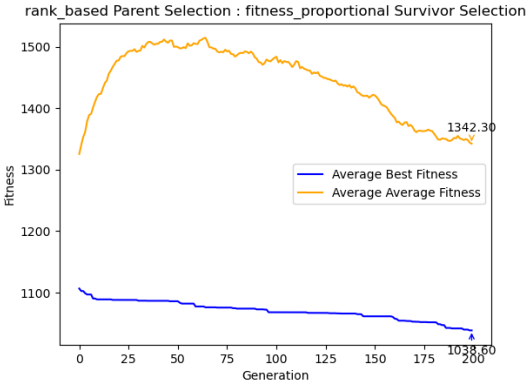
Parent	**Survivor**	**Best Fitness**	**Average Fitness**
Binary	Binary	1219	1220
Binary	Fitness Prop	1228	1676
Binary	Random	1158	1178
Binary	Rank Based	1256	1256
Binary	Truncation	1162	1162
Fitness Prop	Binary	1218	1218
Fitness Prop	Fitness Prop	1298	2709
Fitness Prop	Random	1255	1892
Fitness Prop	Rank Based	1226	1226
Fitness Prop	**Truncation**	**1138**	**1138**
Random	Binary	1221	1221
Random	Fitness Prop	1292	2468
Random	Random	1239	1755
Random	Rank Based	1210	1210
Random	Truncation	1148	1148
Rank Based	Binary	1192	1192
Rank Based	Fitness Prop	1208	1672
Rank Based	Random	1166	1186
Rank Based	Rank Based	1198	1198
Rank Based	Truncation	1170	1170
Truncation	Binary	1276	1276
Truncation	Fitness Prop	1290	1503
Truncation	Random	1284	1294
Truncation	**Rank Based**	**1302**	**1302**
Truncation	Truncation	1266	1266

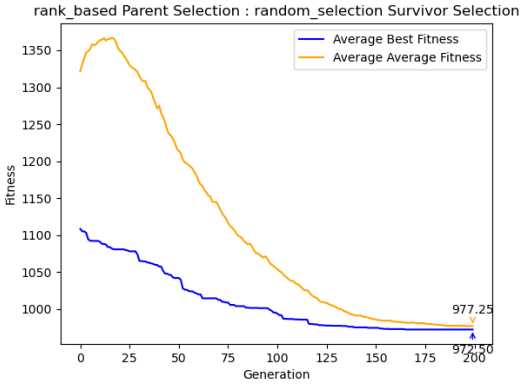
Table 2: Comparison of Parent and Survivor Selection Methods

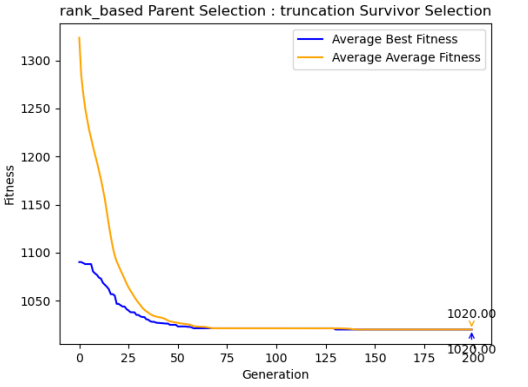
Instance 2 (abz7)

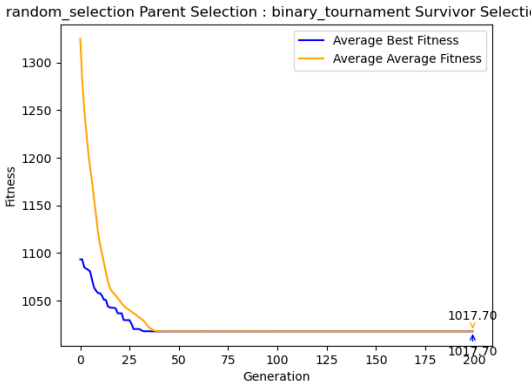


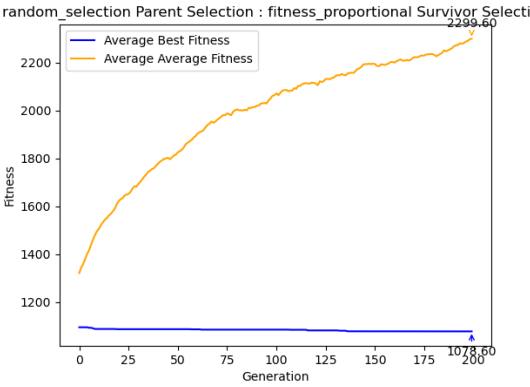

rank_based Parent Selection : binary_tournament Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1025.31
1025.30

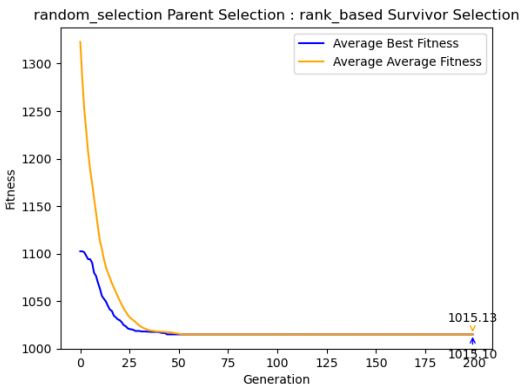

rank_based Parent Selection : fitness_proportional Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1342.30
1038.60

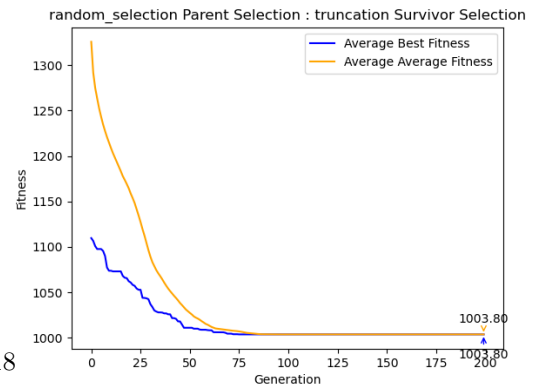

rank_based Parent Selection : random_selection Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
977.25
977.50

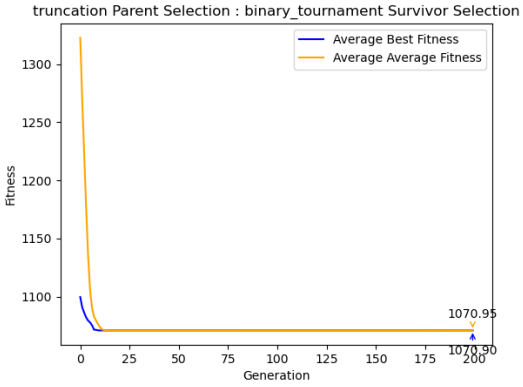

rank_based Parent Selection : truncation Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1020.00
1020.00

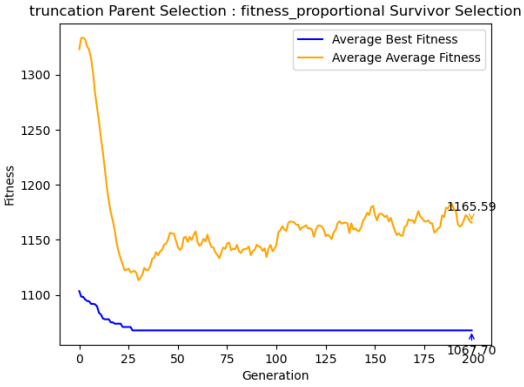

random_selection Parent Selection : binary_tournament Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1017.70
1017.70

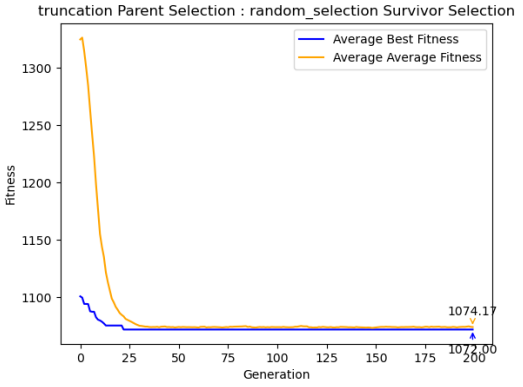

random_selection Parent Selection : fitness_proportional Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
2299.60
1078.60

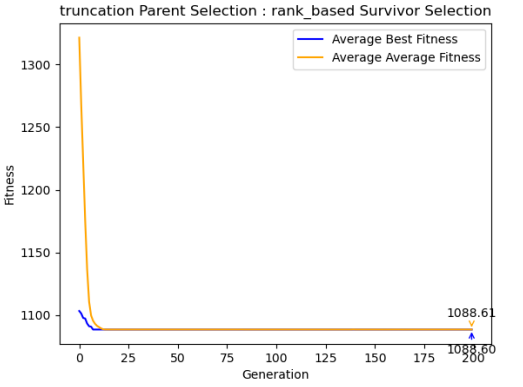

random_selection Parent Selection : rank_based Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1015.13
1015.10


random_selection Parent Selection : truncation Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1003.80
1003.80


truncation Parent Selection : binary_tournament Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1070.95
1070.90


truncation Parent Selection : fitness_proportional Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1165.59
1067.70

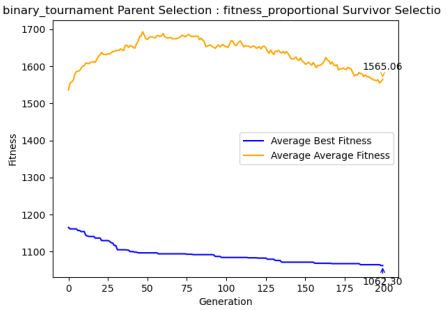

truncation Parent Selection : random_selection Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1074.17
1072.00

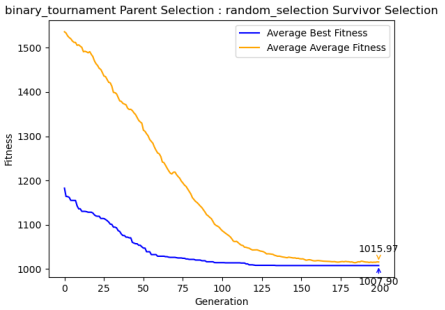

truncation Parent Selection : rank_based Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1088.61
1088.60

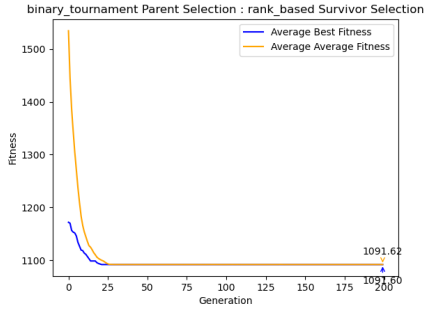
Parent	**Survivor**	**Best Fitness**	**Average Fitness**
Binary	Binary	1024	1024
Binary	Fitness Prop	1028	1321
Binary	Random	974	988
Binary	Rank Based	1024	1024
Binary	Truncation	991	991
Fitness Prop	Binary	1012	1012
Fitness Prop	Fitness Prop	1079	2331
Fitness Prop	Random	1060	1791
Fitness Prop	Rank Based	1021	1021
Fitness Prop	Truncation	987	987
Random	Binary	1017	1017
Random	Fitness Prop	1078	2299
Random	Random	1039	1628
Random	Rank Based	1015	1015
Random	Truncation	1003	1003
Rank Based	Binary	1025	1025
Rank Based	Fitness Prop	1038	1342
Rank Based	**Random**	**972**	**977**
Rank Based	Rank Based	1057	1057
Rank Based	Truncation	1020	1020
Truncation	Binary	1070	1070
Truncation	Fitness Prop	1067	1165
Truncation	Random	1072	1074
Truncation	**Rank Based**	**1088**	**1088**
Truncation	Truncation	1068	1068

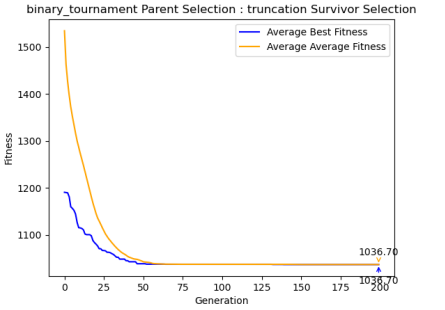
Table 3: Comparison of Parent and Survivor Selection Methods

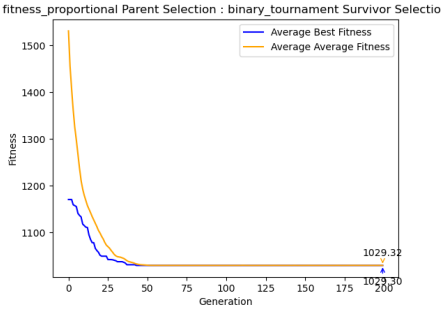
Instance 3 (la19)

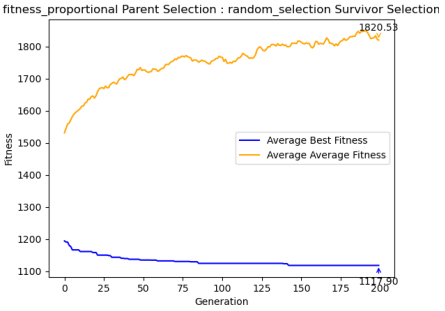


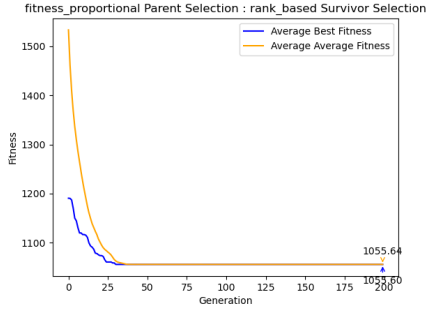


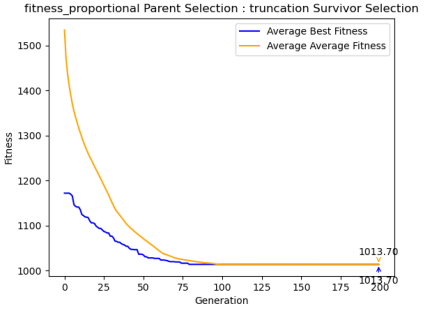


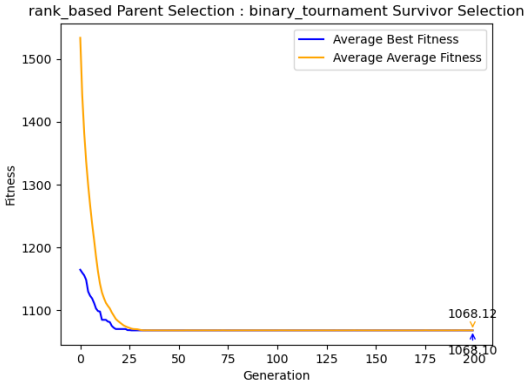


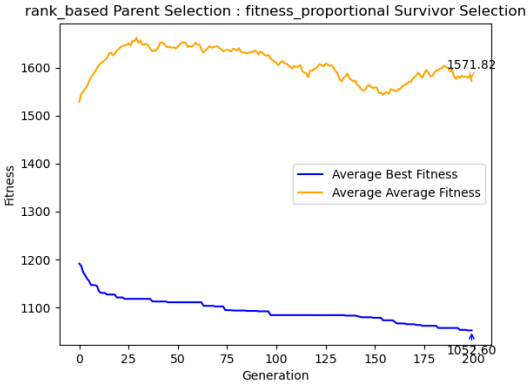


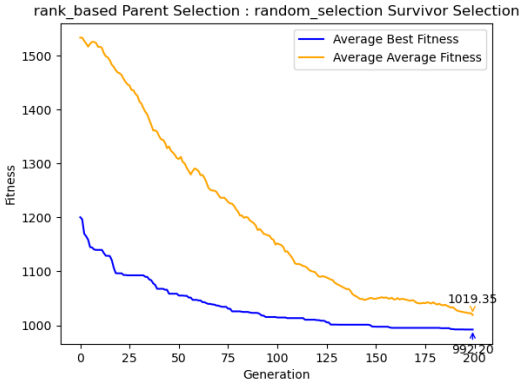


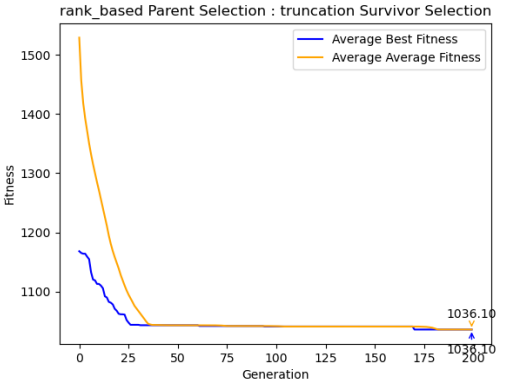


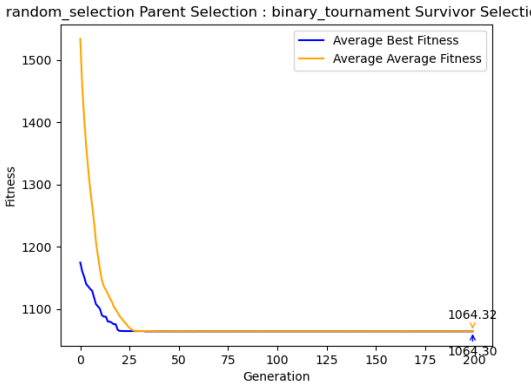


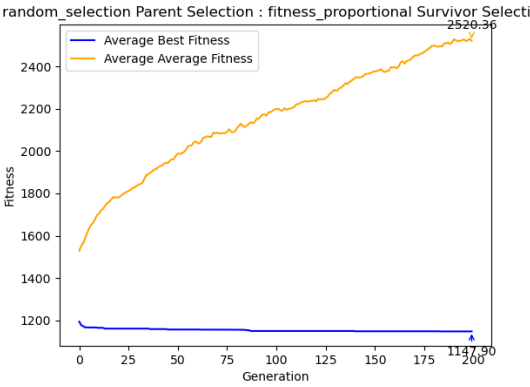

rank_based Parent Selection : binary_tournament Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1068.12
1068.10

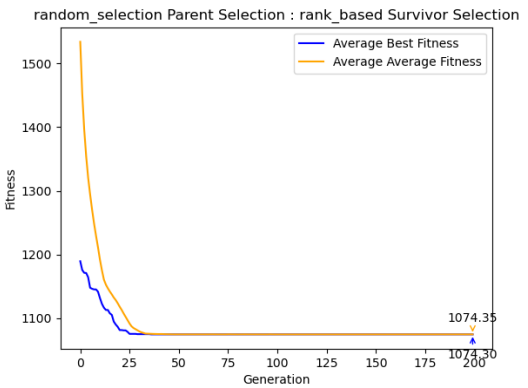

rank_based Parent Selection : fitness_proportional Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1571.82
1052.60

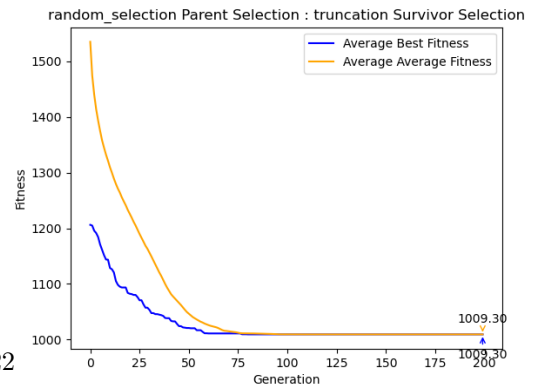

rank_based Parent Selection : random_selection Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1019.35
992.20

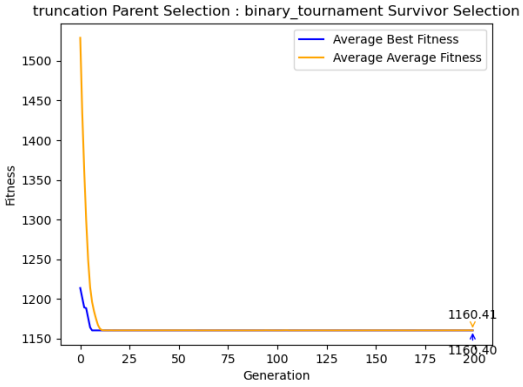

rank_based Parent Selection : truncation Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1036.10
1038.10

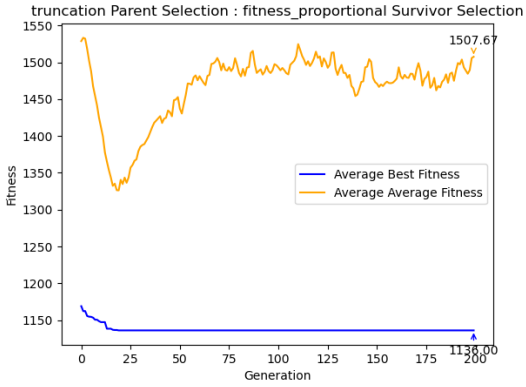

random_selection Parent Selection : binary_tournament Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1064.32
1064.30

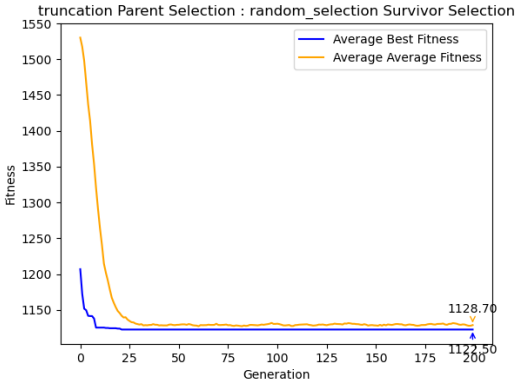

random_selection Parent Selection : fitness_proportional Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
2520.36
1147.90

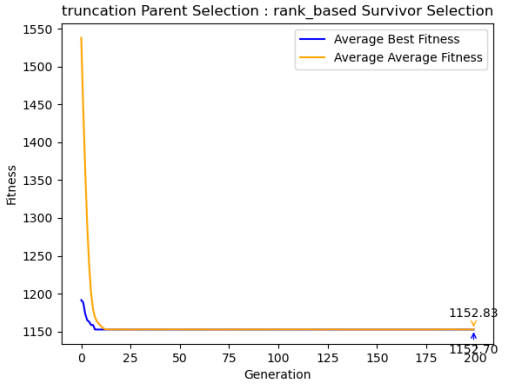

random_selection Parent Selection : rank_based Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1074.35
1074.30


random_selection Parent Selection : truncation Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1009.30
1009.30


truncation Parent Selection : binary_tournament Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1160.41
1160.40


truncation Parent Selection : fitness_proportional Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1507.67
1138.00


truncation Parent Selection : random_selection Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1128.70
1122.50


truncation Parent Selection : rank_based Survivor Selection
Average Best Fitness
Average Average Fitness
Fitness
Generation
1152.83
1152.70

Parent	Survivor	Best Fitness	Average Fitness
Binary	Binary	1077	1077
Binary	Fitness Prop	1062	1565
Binary	Random	1015	1007
Binary	Rank Based	1091	1091
Binary	Truncation	1036	1036
Fitness Prop	Binary	1029	1029
Fitness Prop	Fitness Prop	1146	2714
Fitness Prop	Random	1117	1820
Fitness Prop	Rank Based	1055	1055
Fitness Prop	Truncation	1013	1013
Random	Binary	1064	1064
Random	Fitness Prop	1147	2520
Random	Random	1084	1584
Random	Rank Based	1074	1074
Random	Truncation	1009	1009
Rank Based	Binary	1068	1068
Rank Based	Fitness Prop	1052	1571
Rank Based	**Random**	**992**	**1019**
Rank Based	Rank Based	1040	1040
Rank Based	Truncation	1036	1036
Truncation	**Binary**	**1160**	**1160**
Truncation	Fitness Prop	1138	1507
Truncation	Random	1128	1122
Truncation	Rank Based	1152	1152
Truncation	Truncation	1125	1125

Table 4: Comparison of Parent and Survivor Selection Methods

2.5 Analysis

2.5.1 Random Selection

Using random selection for both parent and, particularly, survivor selection surprisingly resulted in consistently good fitness values across all three instances. This outcome highlights the potential benefits of introducing randomness into the evolutionary process. More notably, random survivor selection appears to play a crucial role in maintaining diversity across generations which has far better results as the search space was big. Thus our algorithm needed to get more explorative as well.

2.5.2 Parent and Survivor Selection Schemes

Other than Random, Truncation, and Rank-based selection schemes also demonstrated impressive performance as both parent and survivor selection strategies. In contrast, Fitness Proportional Selection (FPS) consistently underperformed. Its reliance on fitness-proportional probabilities often led to suboptimal results, likely due to the excessive favoring of individuals with slightly higher fitness, which can result in early convergence and loss of genetic diversity.

2.5.3 Variability in Average Fitness

In the poorly performing combinations, particularly those involving Fitness Proportional Selection (FPS), there was significant variability in the average fitness values. This instability indicated a failure to consistently converge toward optimal solutions. One possible reason for this behavior is the inherent bias of FPS toward selecting individuals with higher fitness early on, which can prematurely narrow the search space. As a result, the algorithm may miss exploring potentially better solutions in less explored areas of the search space.

2.5.4 Different Configurations of Parameters

Adjusting the parameters, particularly by increasing the population size and number of generations, led to significantly better results during optimization. A larger population size enhanced diversity, providing a broader pool of candidate solutions and improving the exploration of the search space. This resulted in a stronger initial population to work with. Additionally, increasing the number of generations allowed the population more time to evolve and converge toward optimal solutions, further improving overall performance.

2.5.5 Conclusion

Across the three instances, due to the vast search space, schemes like Random and Truncation often outperformed traditional approaches like FPS for both parent and survivor selection. Additionally, there was significant variability in the results when running the same combination of schemes repeatedly, which can be attributed to the extensive search space the evolutionary algorithm had to navigate. It was also observed that the algorithm frequently converged to suboptimal solutions, getting trapped in local minima. This issue was only resolved by restarting the code and rerunning the optimization process.

2.6 Optimal Solution

After optimizing using all the combinations of selection schemes we chose the best results and further optimized our results by changing the parameters and increasing our generations. These are the results for the three instances of JSSP:

Parameters Used for Best Optimization:

```
population_size = 500
num_generations = 1000
mutation_rate = 1.0
num_offspring = 70
num_iterations = 1
```

Instance 1 (ft10)
Best Fitness: = **1080**

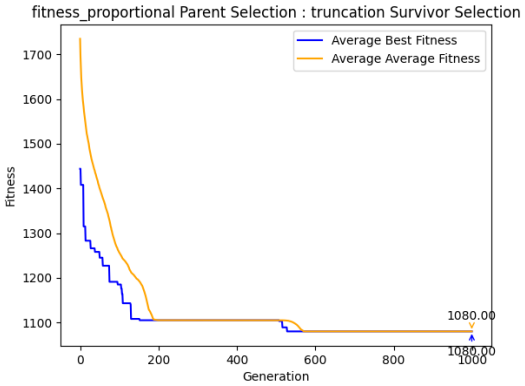


Figure 14: Optimal Solution - ft10

Instance 2 (abz7)
Best Fitness: = **887**

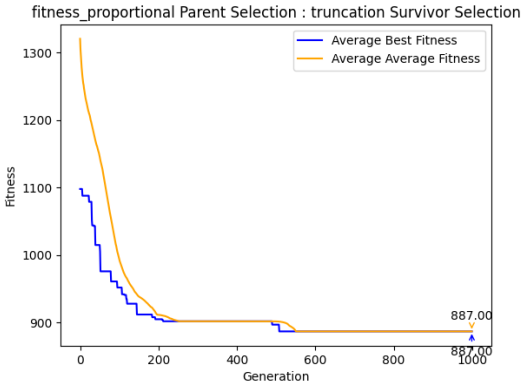


Figure 15: Optimal Solution - abz7

Instance 3 (la19)
Best Fitness: = **992**

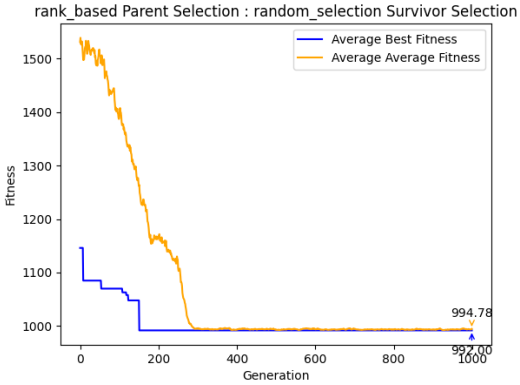


Figure 16: Optimal Solution - la19

3 Mona Lisa problem

3.1 Chromosome representation

For this specific problem, the chromosomes were individual drawings trying to depict Mona Lisa, each made up of 50 semi-transparent polygons, as was described in the original blog[1]. Each polygon has two attributes, namely its vertices, its color (including alpha for transparency), as well as which region of the image the polygon should cover. The number of vertices per polygon was between 3 and 10, which was randomly decided.

3.2 Fitness function

As is for every genetic algorithm, the fitness function evaluates how close a member of the population is to the desired result. In this specific case, we had to consider both the polygon color. Initially, the plan was to make use of Euclidean difference, denoted by:

$$\text{ED} = \sum_{i=1}^{n} \sqrt{(R_i - R_i')^2 + (G_i - G_i')^2 + (B_i - B_i')^2}$$

However, the disadvantage of this, as noted in this Medium article[2], is that the difference in color measured by Euclidean distance is not an accurate portrayal of how our eyes perceive the difference between two colors. Instead, we used the Delta E (CIE76) formula, which is defined as:

$$\Delta E = \sqrt{(L_1 - L_2)^2 + (a_1 - a_2)^2 + (b_1 - b_2)^2}$$

where L, a, b represent the lightness and chromatic components of the CIELAB color space[3]. This metric is a more accurate representation of human perception of color differences, making it a more effective fitness function for our evolutionary image reconstruction.

Note that in this case, lower fitness values indicate better similarity to the target image.

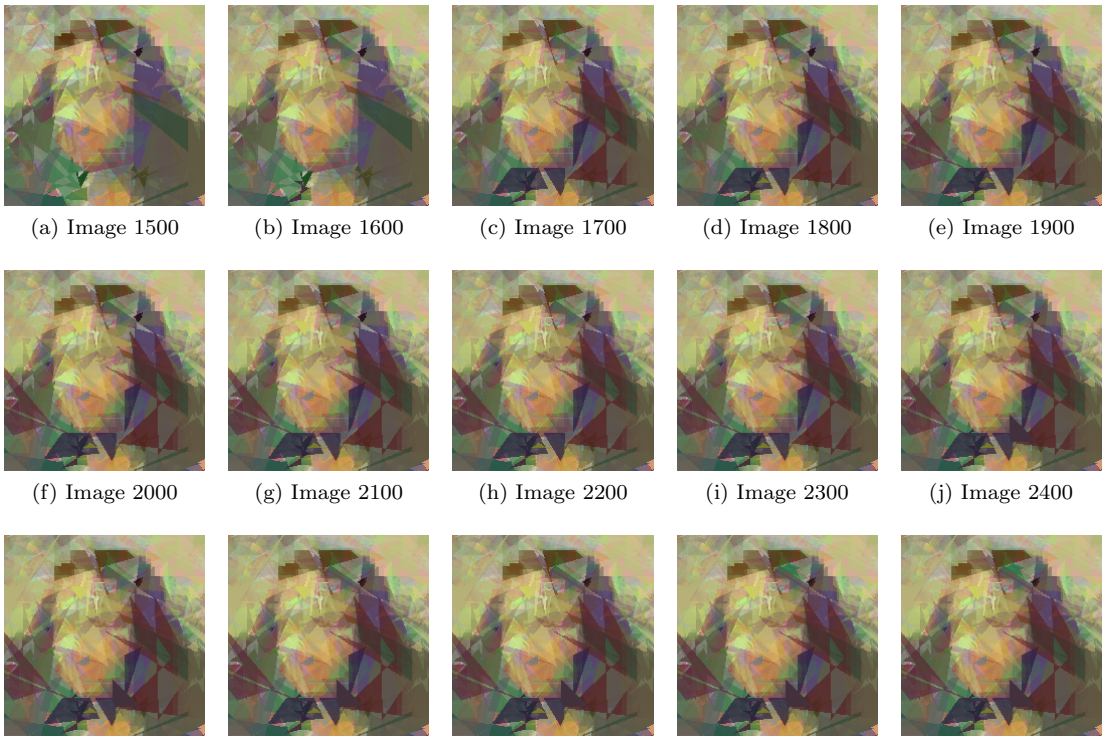
3.3 Images

(a) Image 0 (b) Image 100 (c) Image 200 (d) Image 300 (e) Image 400

(f) Image 500 (g) Image 600 (h) Image 700 (i) Image 800 (j) Image 900

(k) Image 1000 (l) Image 1100 (m) Image 1200 (n) Image 1300 (o) Image 1400

Figure 17: Fittest Images 0-1400 in Jumps of 100

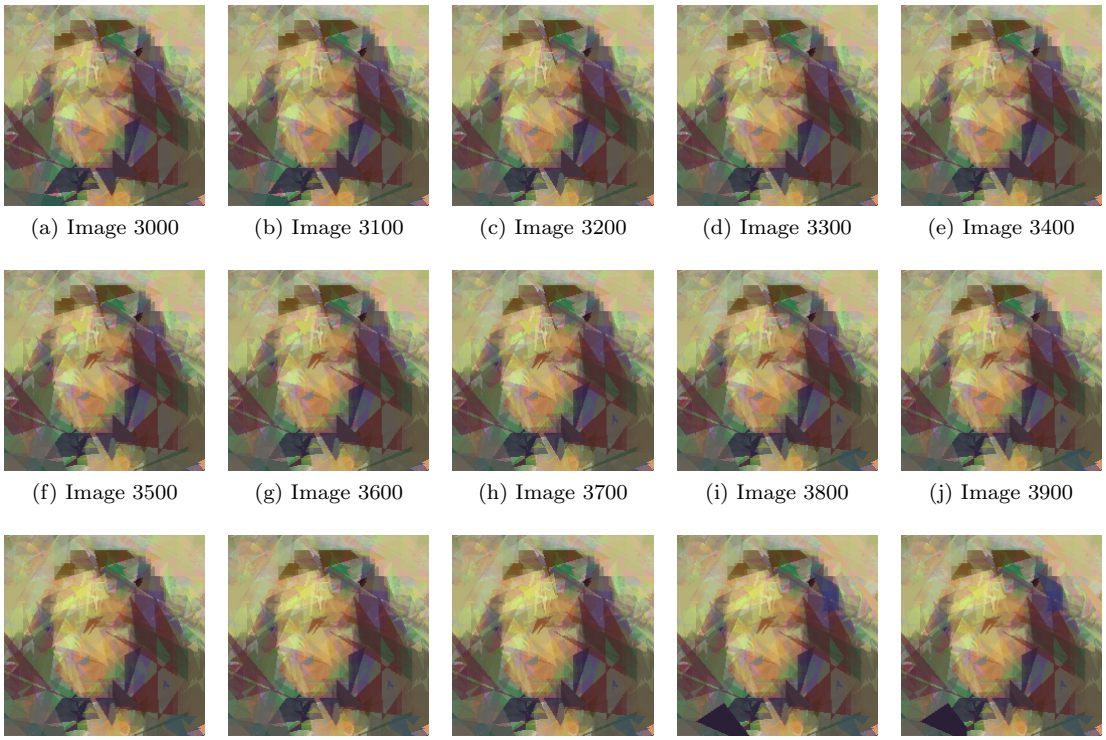


(a) Image 1500 (b) Image 1600 (c) Image 1700 (d) Image 1800 (e) Image 1900

(f) Image 2000 (g) Image 2100 (h) Image 2200 (i) Image 2300 (j) Image 2400

(k) Image 2500 (l) Image 2600 (m) Image 2700 (n) Image 2800 (o) Image 2900

Figure 18: Fittest Images 1500-2900 in Jumps of 100

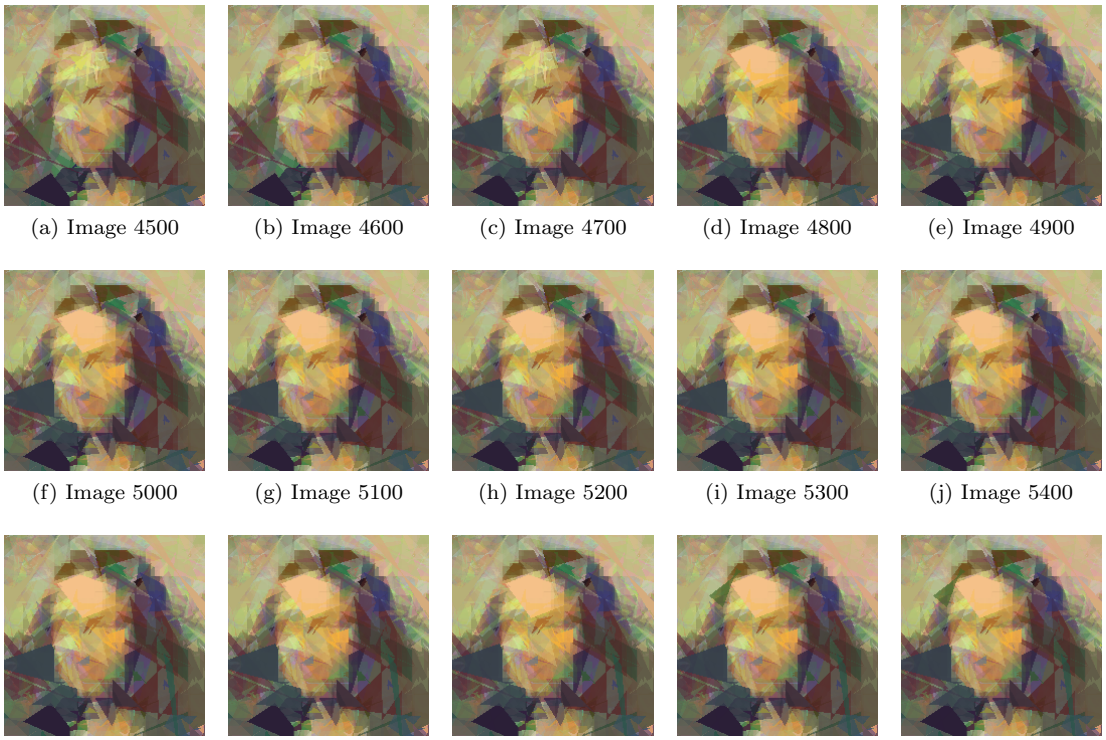


(a) Image 3000 (b) Image 3100 (c) Image 3200 (d) Image 3300 (e) Image 3400

(f) Image 3500 (g) Image 3600 (h) Image 3700 (i) Image 3800 (j) Image 3900

(k) Image 4000 (l) Image 4100 (m) Image 4200 (n) Image 4300 (o) Image 4400

Figure 19: Fittest Images 3000-4400 in Jumps of 100

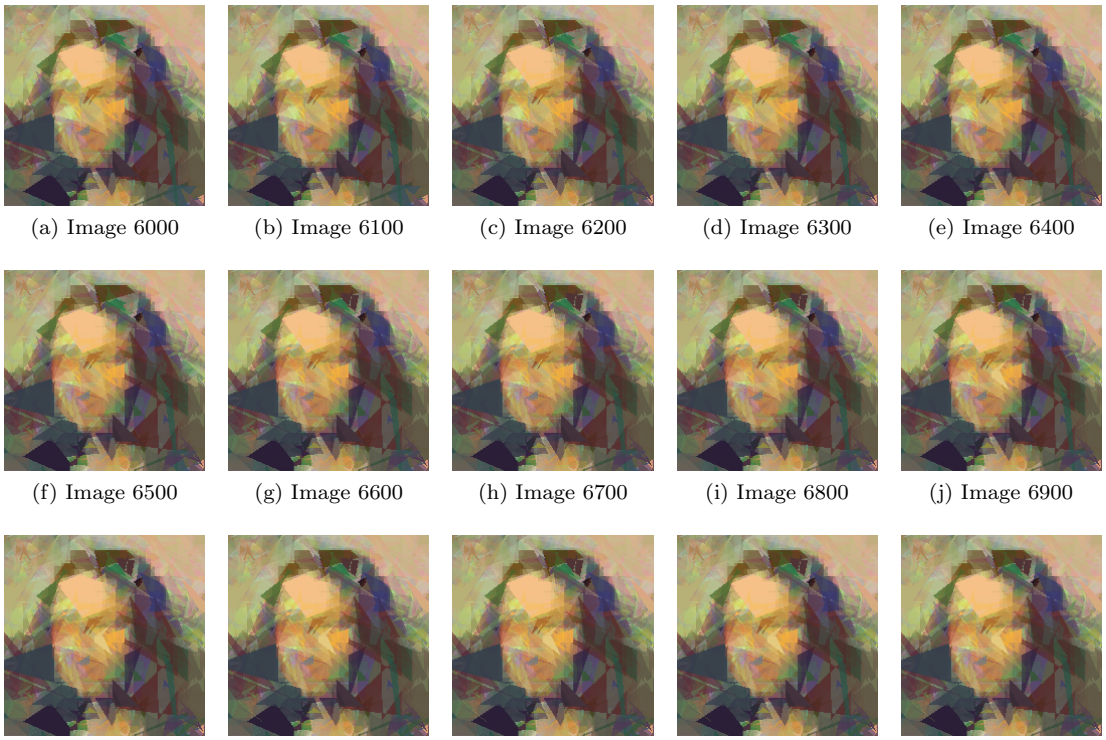


(a) Image 4500 (b) Image 4600 (c) Image 4700 (d) Image 4800 (e) Image 4900

(f) Image 5000 (g) Image 5100 (h) Image 5200 (i) Image 5300 (j) Image 5400

(k) Image 5500 (l) Image 5600 (m) Image 5700 (n) Image 5800 (o) Image 5900

Figure 20: Fittest Images 4500-5900 in Jumps of 100

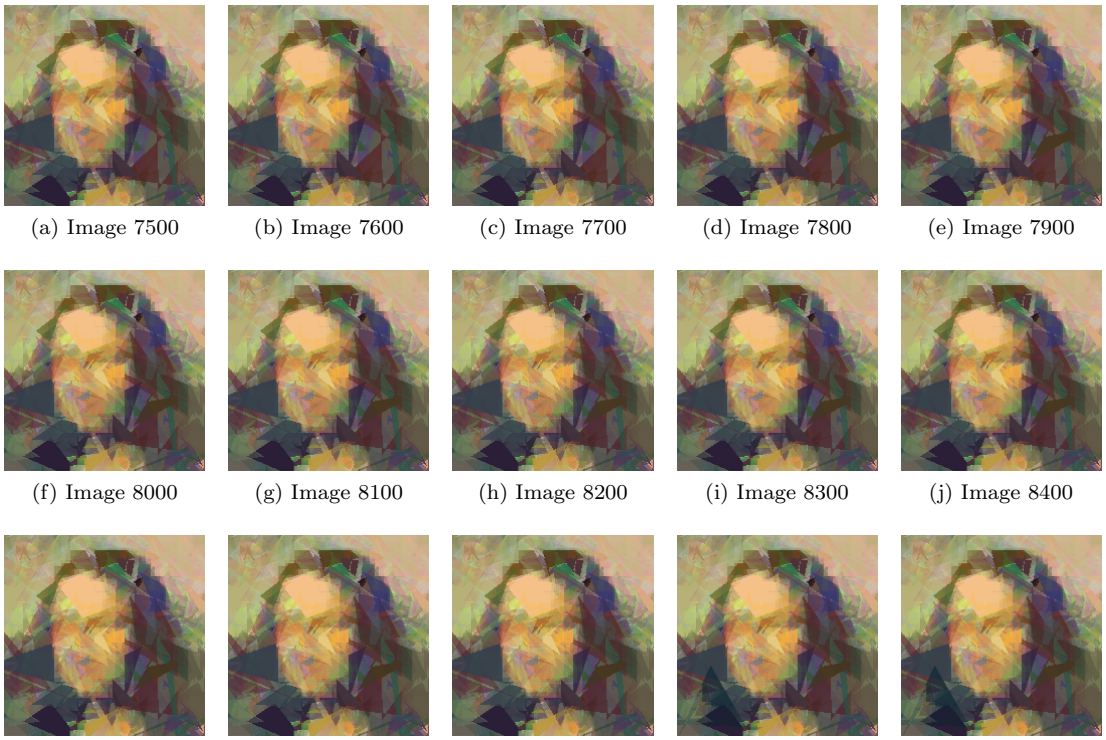


(a) Image 6000 (b) Image 6100 (c) Image 6200 (d) Image 6300 (e) Image 6400

(f) Image 6500 (g) Image 6600 (h) Image 6700 (i) Image 6800 (j) Image 6900

(k) Image 7000 (l) Image 7100 (m) Image 7200 (n) Image 7300 (o) Image 7400

Figure 21: Fittest Images 6000-7400 in Jumps of 100

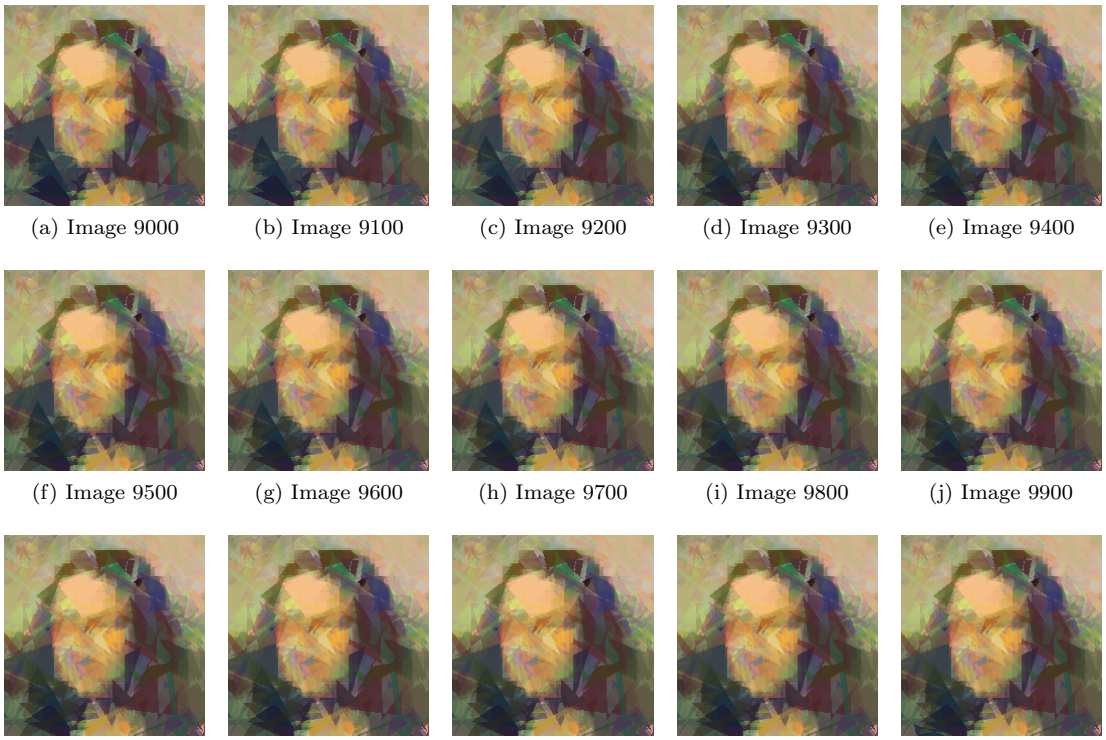


(a) Image 7500 (b) Image 7600 (c) Image 7700 (d) Image 7800 (e) Image 7900

(f) Image 8000 (g) Image 8100 (h) Image 8200 (i) Image 8300 (j) Image 8400

(k) Image 8500 (l) Image 8600 (m) Image 8700 (n) Image 8800 (o) Image 8900

Figure 22: Fittest Images 7500-8900 in Jumps of 100



(a) Image 9000 (b) Image 9100 (c) Image 9200 (d) Image 9300 (e) Image 9400

(f) Image 9500 (g) Image 9600 (h) Image 9700 (i) Image 9800 (j) Image 9900

(k) Image 10000 (l) Image 10100 (m) Image 10200 (n) Image 10300 (o) Image 10400

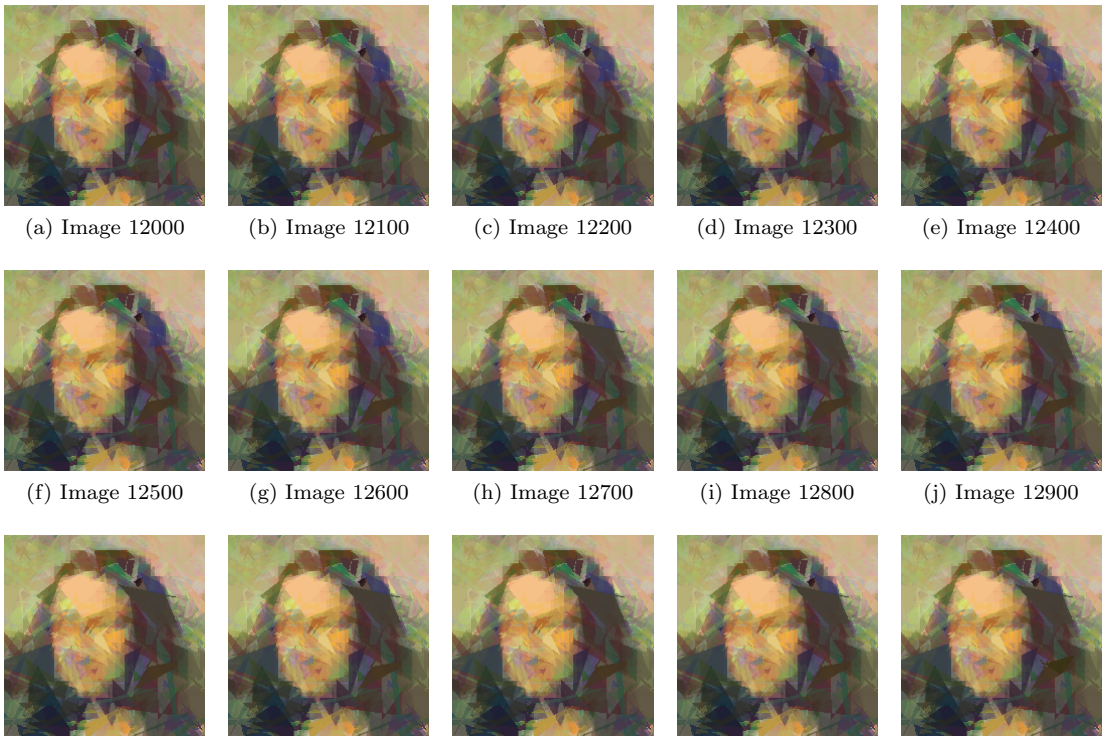
Figure 23: Fittest Images 9000-10400 in Jumps of 100

(a) Image 10500 (b) Image 10600 (c) Image 10700 (d) Image 10800 (e) Image 10900

(f) Image 11000 (g) Image 11100 (h) Image 11200 (i) Image 11300 (j) Image 11400

(k) Image 11500 (l) Image 11600 (m) Image 11700 (n) Image 11800 (o) Image 11900

Figure 24: Fittest Images 10500-11900 in Jumps of 100



(a) Image 12000 (b) Image 12100 (c) Image 12200 (d) Image 12300 (e) Image 12400

(f) Image 12500 (g) Image 12600 (h) Image 12700 (i) Image 12800 (j) Image 12900

(k) Image 13000 (l) Image 13100 (m) Image 13200 (n) Image 13300 (o) Image 13400

Figure 25: Fittest Images 12000-13400 in Jumps of 100


(a) Image 13500


(b) Image 13600


(c) Image 13700


(d) Image 13800


(e) Image 13900


(f) Image 14000


(g) Image 14100


(h) Image 14200


(i) Image 14300


(j) Image 14400


(k) Image 14500


(l) Image 14600


(m) Image 14700


(n) Image 14800


(o) Image 14900

Figure 26: Final Fittest Image at Generation 14999

3.4 Analysis and Findings

In the buildup to coming up with the final solution, we experimented with various settings and configurations to determine the most effective approach for evolving the image. Our findings are as follows:

- **Selection Schemes:** Tournament selection consistently outperformed random selection by maintaining population diversity while favoring better solutions.
- **Crossover Strategies:** Region-based crossover maintained structural integrity better than uniform blending, leading to quicker convergence. Uniform blending is a crossover method where pixel values from two parent images are averaged using a random weight (say 3:7 ratio Parent 1 to Parent 2), often resulting in a smooth but less structurally diverse offspring.
- **Mutation Effects:** A *medium* mutation rate (0.3) prevented premature convergence thereby allowing sufficient exploration of the solution space.
- **Population Size:** Larger populations (100+) resulted in smoother fitness progression but increased computation time exponentially. In our personal experience, it took 4 hours just to generate 6000 generations.

Overall, the combination of tournament selection, region-based crossover, and a balanced mutation rate yielded the best results in terms of both fitness improvement and visual quality.

3.5 Challenges and Improvements

- **Polygon Limitation**: Higher polygon counts improve accuracy but increase computation time. Similarly, for more vertices per polygon.
- **Dynamic Mutation Rate**: Adaptive mutation rates could prevent premature convergence.
- **Parallel Processing**: GPU acceleration can enhance performance. Running on a 4 core CPU was a very daunting task.

4 Conclusion

This project successfully demonstrates how **Evolutionary Algorithms** can recreate complex images like the Mona Lisa through iterative refinement. While the final image captures the essence of the target, tweaks in the crossover strategies and polygon representation could enhance the quality of the results.

Sebastian Charmot's Medium blog was a huge help in this task.

References

[1] Roger Johansson, *Genetic Programming: Evolution of Mona Lisa*. Blog Post, 2008. Available at: https://rogerjohansson.blog/2008/12/07/genetic-programming-evolution-of-mona-lisa/

[2] Sebastian Charmot, *A True Genetic Algorithm for Image Recreation — Painting the Mona Lisa*. Medium, September 5, 2022. Available at: https://medium.com/@sebastian.charmot/genetic-algorithm-for-image-recreation-4ca546454aaa

[3] Wikipedia contributors, *Color difference*, Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/wiki/Color_difference#CIE76, Last accessed: February 7, 2025.