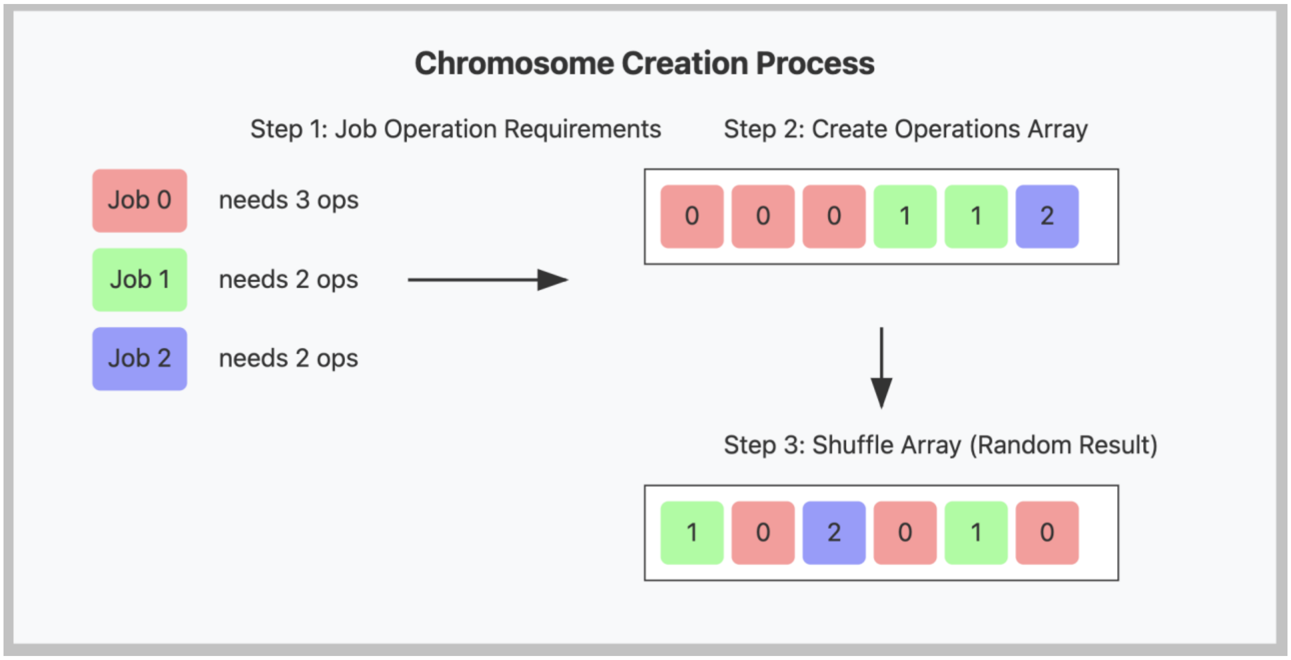
Neural and Evolutionary Computation (NEC)

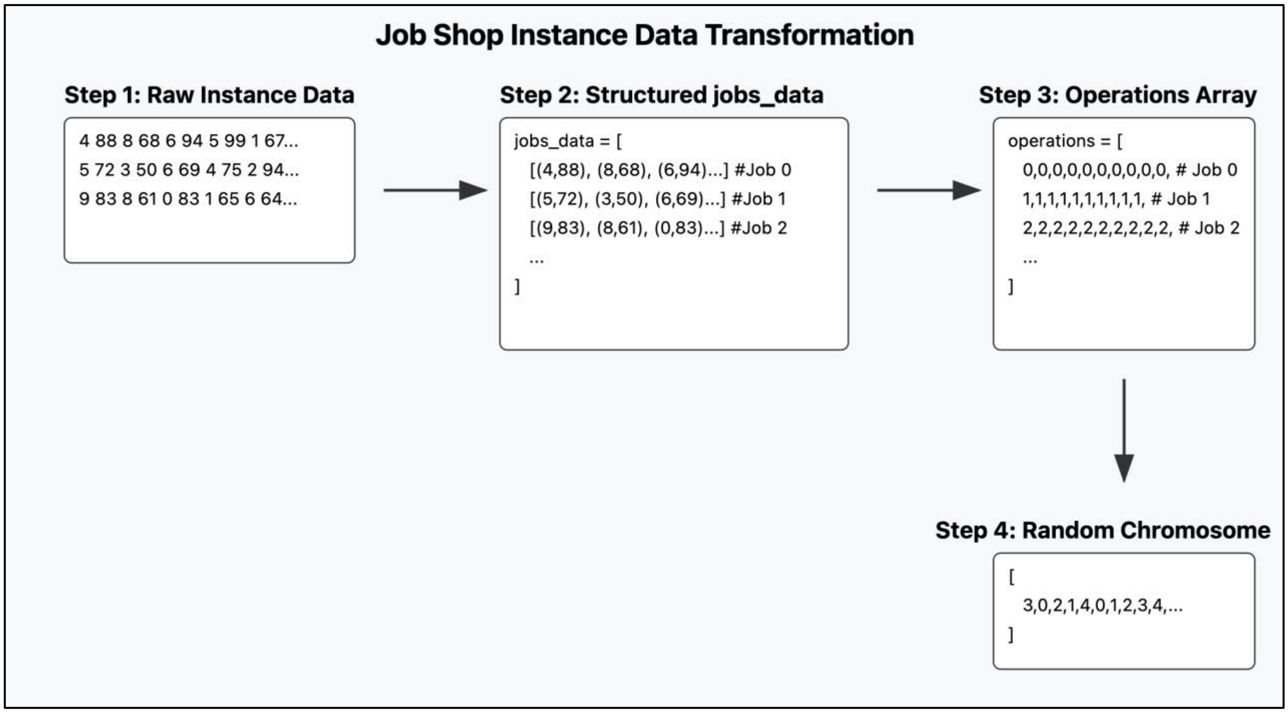
A2: Optimization with Genetic Algorithms
The Job Shop Problem or the Job-Shop Scheduling Problem

Peana Florin Cosmin

GitHub Repo: https://github.com/FlorinCP/JSSP

1. Chromosome Representation and Problem Translation:


Chromosome Creation Process
Step 1: Job Operation Requirements
Step 2: Create Operations Array
Job 0
needs 3 ops
Job 1
needs 2 ops
Job 2
needs 2 ops
0 0 0 1 1 2
Step 3: Shuffle Array (Random Result)
1 0 2 0 1 0

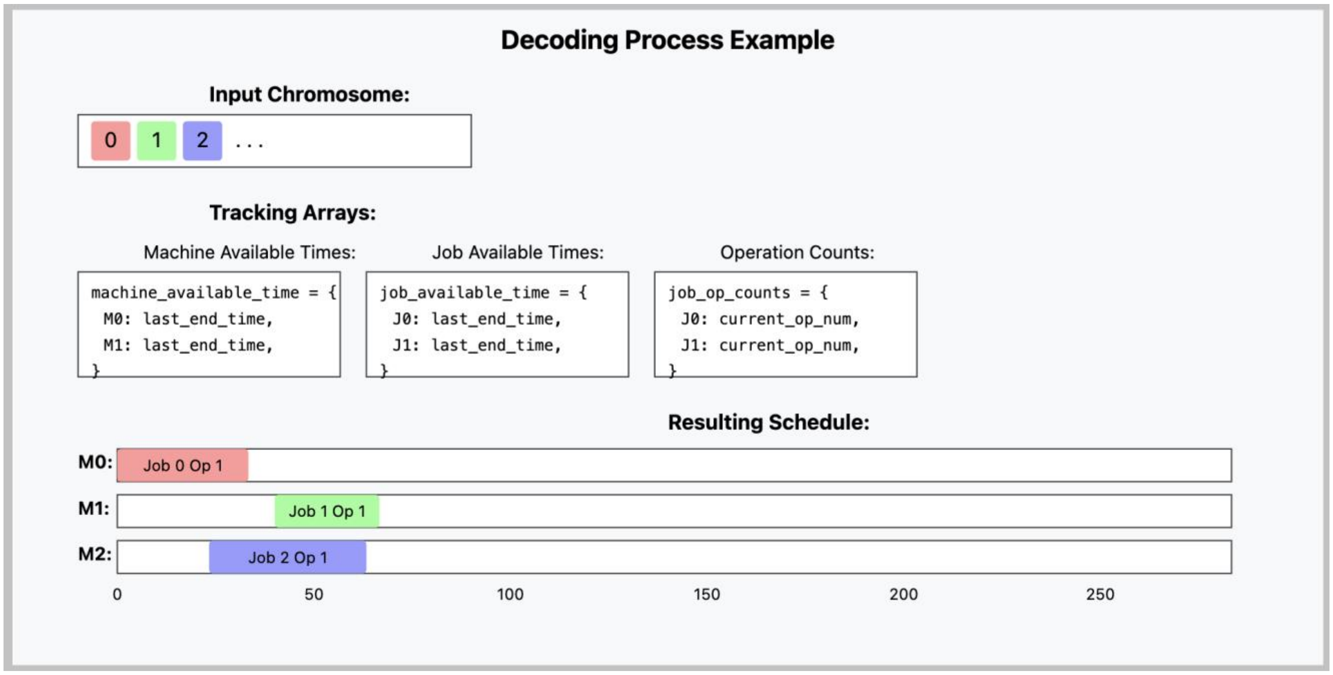

Job Shop Instance Data Transformation
Step 1: Raw Instance Data
Step 2: Structured jobs_data
Step 3: Operations Array
Step 4: Random Chromosome

```
4 88 8 68 6 94 5 99 1 67...
5 72 3 50 6 69 4 75 2 94...
9 83 8 61 0 83 1 65 6 64...
```

```
jobs_data = [
  [(4,88), (8,68), (6,94)...] #Job 0
  [(5,72), (3,50), (6,69)...] #Job 1
  [(9,83), (8,61), (0,83)...] #Job 2
  ...
]
```

```
operations = [
  0,0,0,0,0,0,0,0,0,0, # Job 0
  1,1,1,1,1,1,1,1,1,1, # Job 1
  2,2,2,2,2,2,2,2,2,2, # Job 2
  ...
]
```

```
[
  3,0,2,1,4,0,1,2,3,4,...
]
```



Decoding Process Example

Input Chromosome:

0 1 2 . . .

Tracking Arrays:

Machine Available Times:

```
machine_available_time = {
  M0: last_end_time,
  M1: last_end_time,
}
```

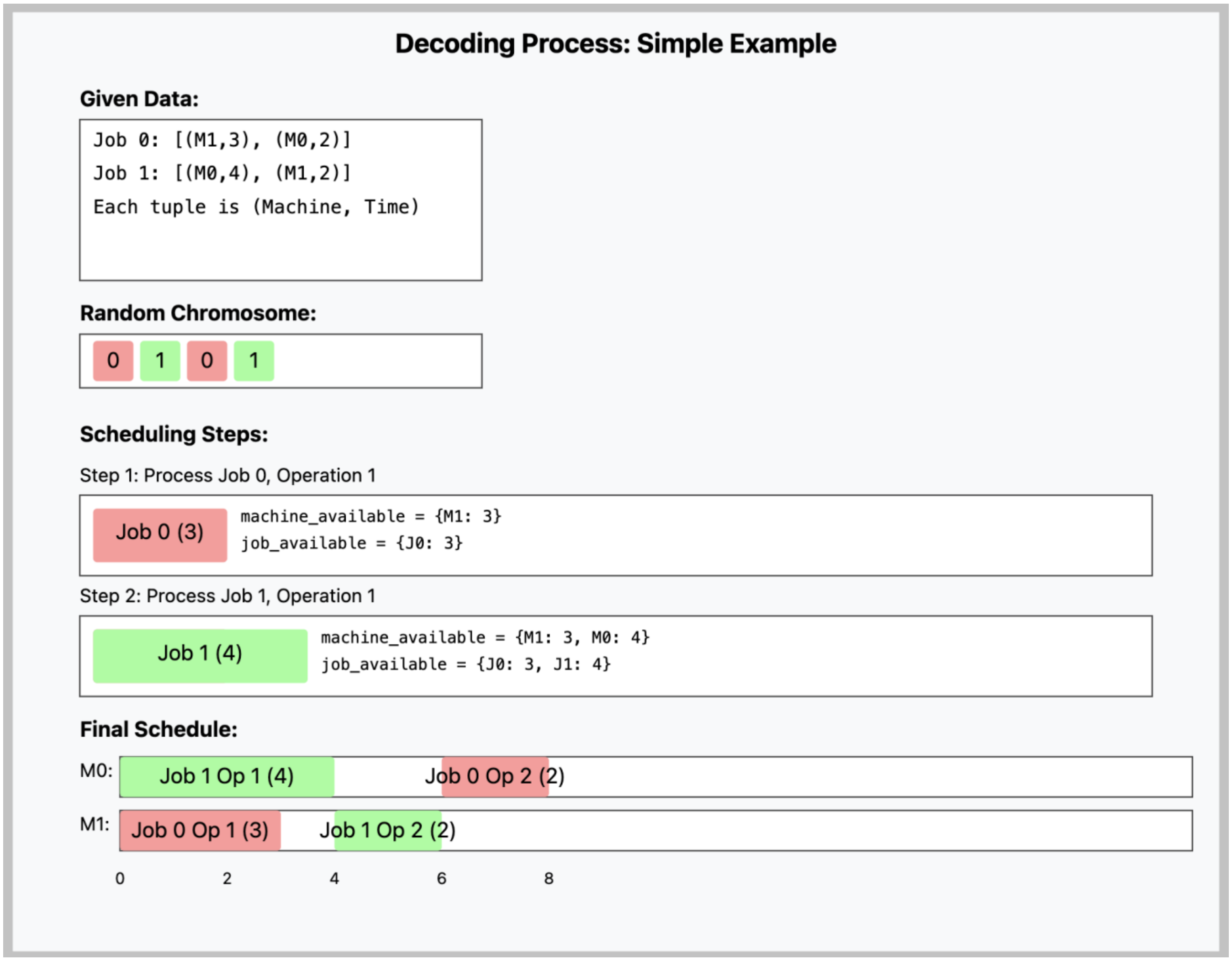
Job Available Times:

```
job_available_time = {
  J0: last_end_time,
  J1: last_end_time,
}
```

Operation Counts:

```
job_op_counts = {
  J0: current_op_num,
  J1: current_op_num,
}
```

Resulting Schedule:



Decoding Process: Simple Example

Given Data:

```
Job 0: [(M1,3), (M0,2)]
Job 1: [(M0,4), (M1,2)]
Each tuple is (Machine, Time)
```

Random Chromosome:

0 1 0 1

Scheduling Steps:

Step 1: Process Job 0, Operation 1

Job 0 (3)

```
machine_available = {M1: 3}
job_available = {J0: 3}
```

Step 2: Process Job 1, Operation 1

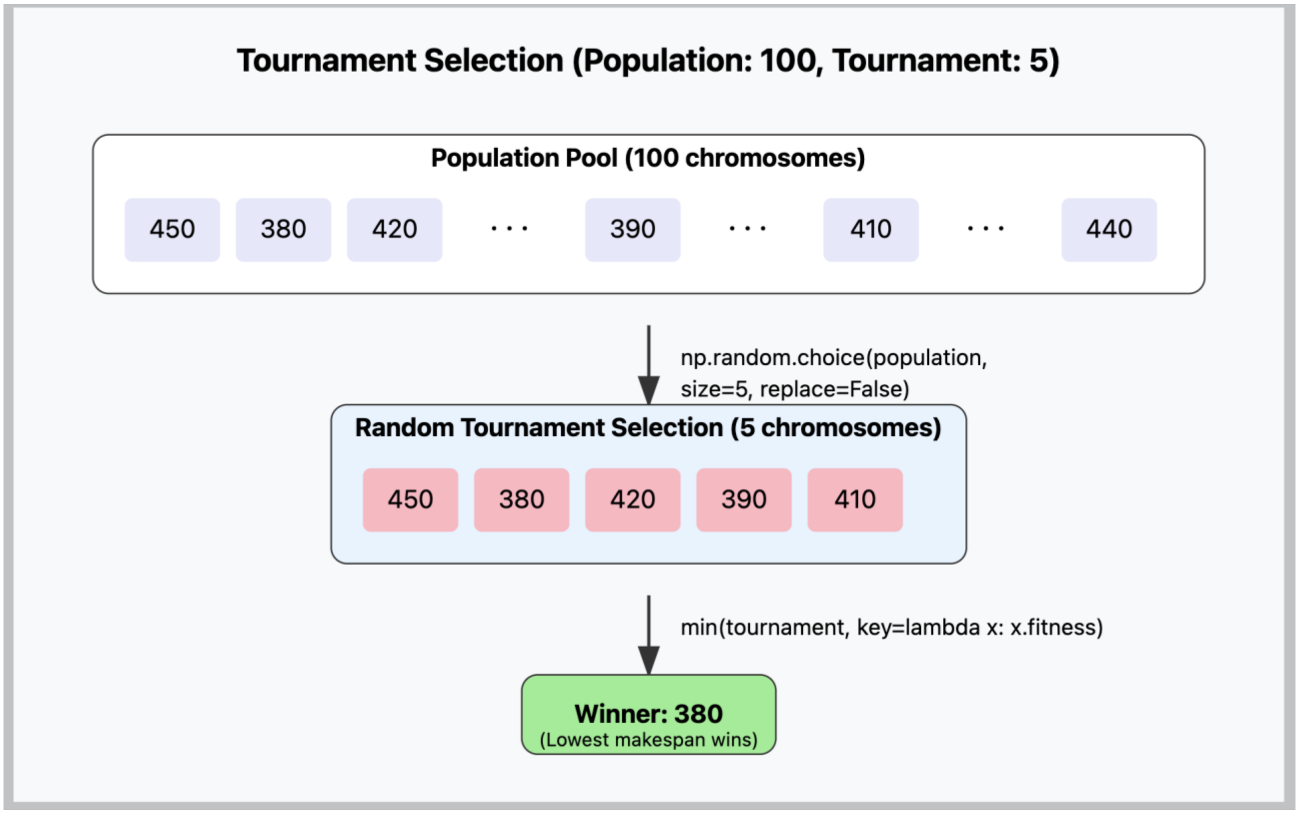
Job 1 (4)

```
machine_available = {M1: 3, M0: 4}
job_available = {J0: 3, J1: 4}
```

Final Schedule:

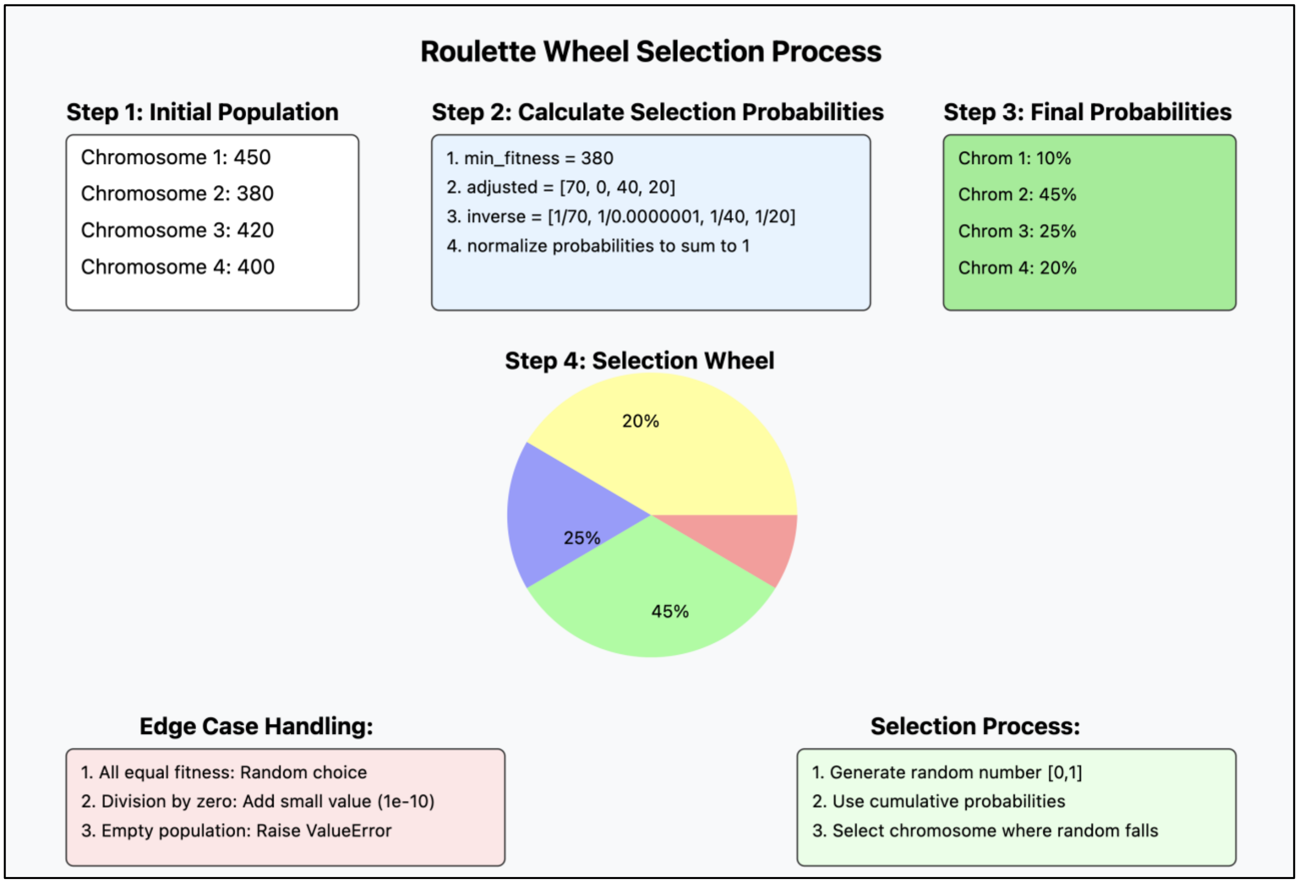
Selection Methods:

1. Tournament Selection:



Tournament Selection (Population: 100, Tournament: 5)
Population Pool (100 chromosomes)
450
380
420
. . .
390
. . .
410
. . .
440
np.random.choice(population, size=5, replace=False)
Random Tournament Selection (5 chromosomes)
450
380
420
390
410
min(tournament, key=lambda x: x.fitness)
Winner: 380
(Lowest makespan wins)

2. Roulette Wheel Selection:



Roulette Wheel Selection Process
Step 1: Initial Population
Step 2: Calculate Selection Probabilities
Step 3: Final Probabilities
Step 4: Selection Wheel
20%
25%
45%
Edge Case Handling:
Selection Process:

Step 1: Initial Population

Chromosome 1: 450
Chromosome 2: 380
Chromosome 3: 420
Chromosome 4: 400

Step 2: Calculate Selection Probabilities

1. min_fitness = 380
2. adjusted = [70, 0, 40, 20]
3. inverse = [1/70, 1/0.0000001, 1/40, 1/20]
4. normalize probabilities to sum to 1

Step 3: Final Probabilities

Chrom 1: 10%
Chrom 2: 45%
Chrom 3: 25%
Chrom 4: 20%

Edge Case Handling:

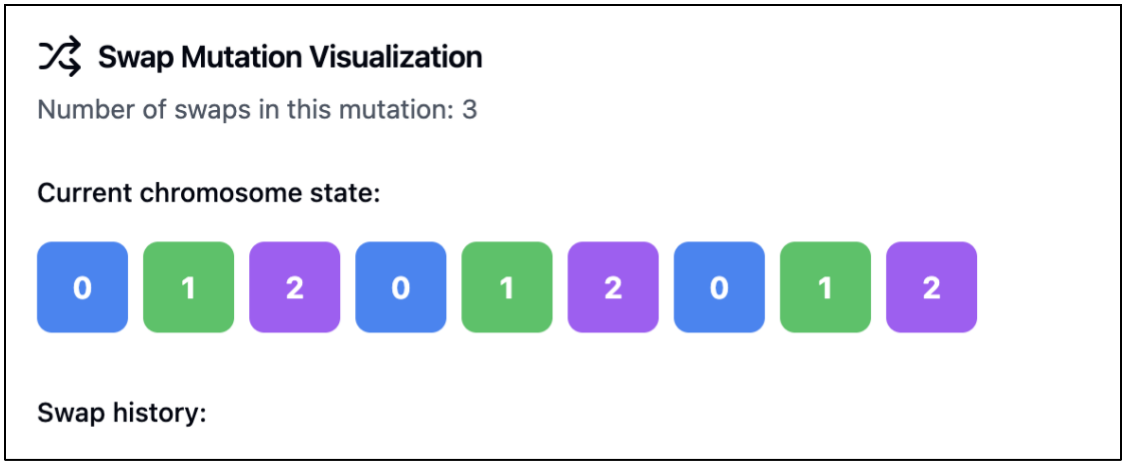
1. All equal fitness: Random choice
2. Division by zero: Add small value (1e-10)
3. Empty population: Raise ValueError

Selection Process:

1. Generate random number [0,1]
2. Use cumulative probabilities
3. Select chromosome where random falls

Mutation Methods:

1. Swap Mutation:



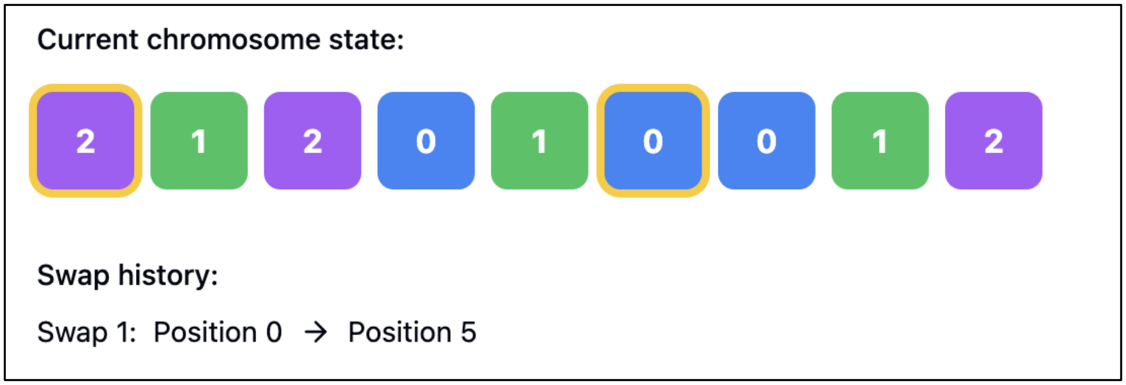
Swap Mutation Visualization

Number of swaps in this mutation: 3

Current chromosome state:

0	1	2	0	1	2	0	1	2

Swap history:

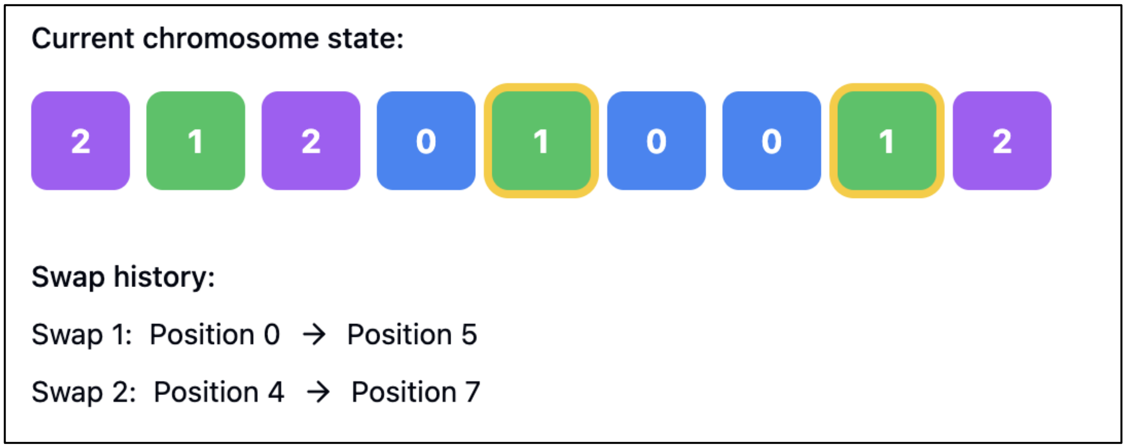


Current chromosome state:

2	1	2	0	1	0	0	1	2

Swap history:

Swap 1: Position 0 → Position 5



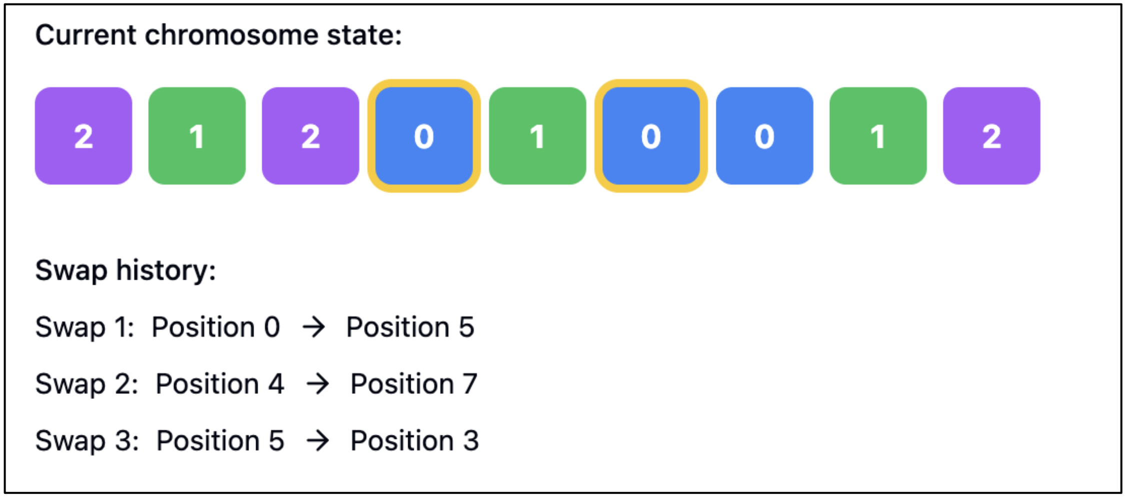
Current chromosome state:

2	1	2	0	1	0	0	1	2

Swap history:

Swap 1: Position 0 → Position 5

Swap 2: Position 4 → Position 7



Current chromosome state:

2	1	2	0	1	0	0	1	2

Swap history:

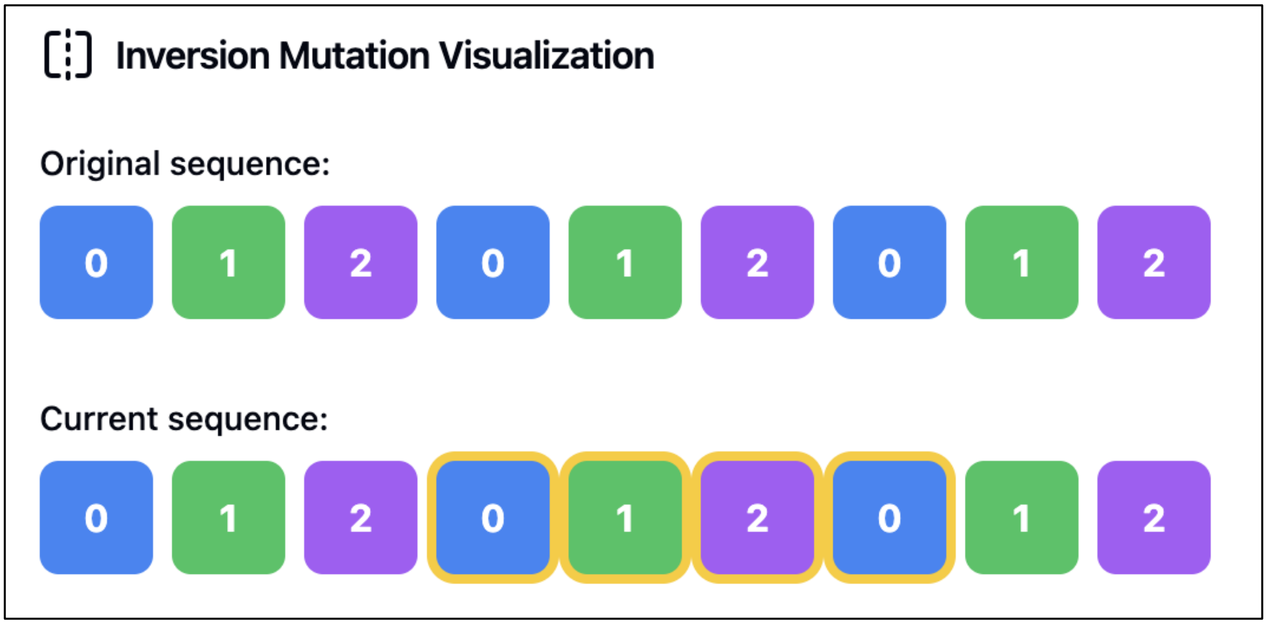
Swap 1: Position 0 → Position 5

Swap 2: Position 4 → Position 7

Swap 3: Position 5 → Position 3

2. Inversion Mutation:

Before mutation, with the indexes for inversion selected.



Inversion Mutation Visualization

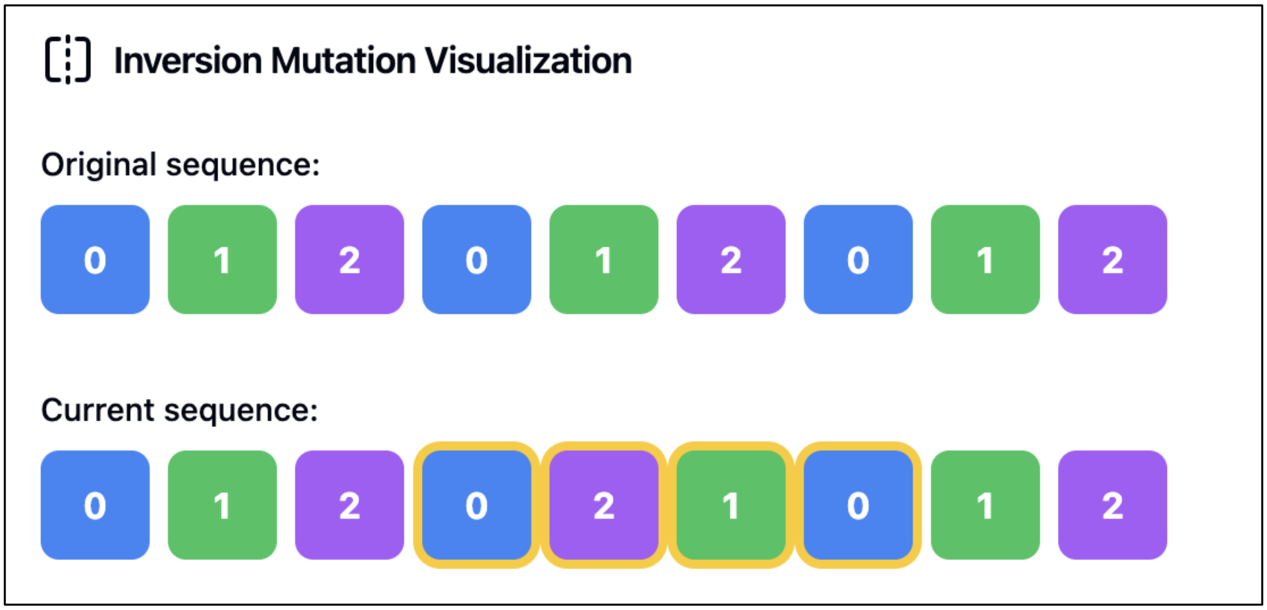
Original sequence:

0 1 2 0 1 2 0 1 2

Current sequence:

0 1 2 0 1 2 0 1 2

After mutation, the elements from the prev. selected indexes were reversed.



Inversion Mutation Visualization

Original sequence:

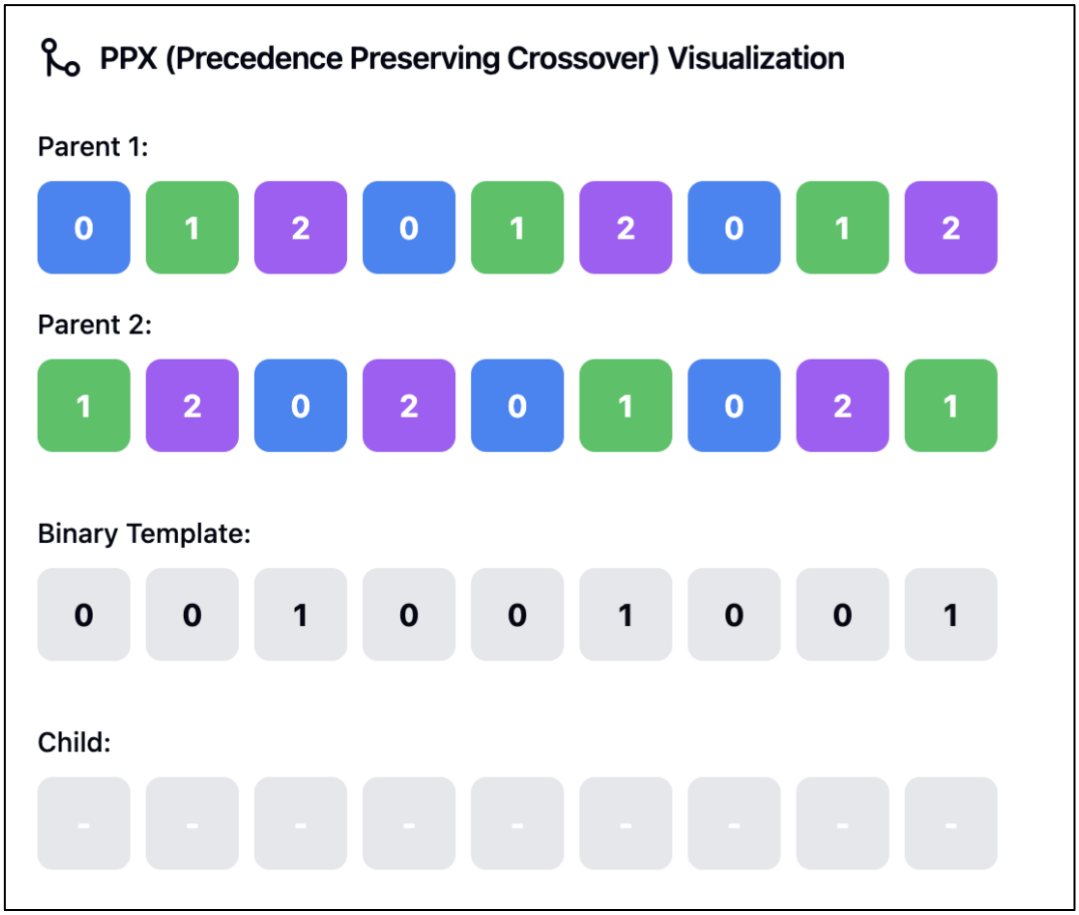
0 1 2 0 1 2 0 1 2

Current sequence:

0 1 2 0 2 1 0 1 2

Crossover Methods:

1. PPX (Precedence Preserving Crossover) [1 – Parent 1, 0 – Parent 2] [Convention]



PPX (Precedence Preserving Crossover) Visualization

Parent 1:

0	1	2	0	1	2	0	1	2

Parent 2:

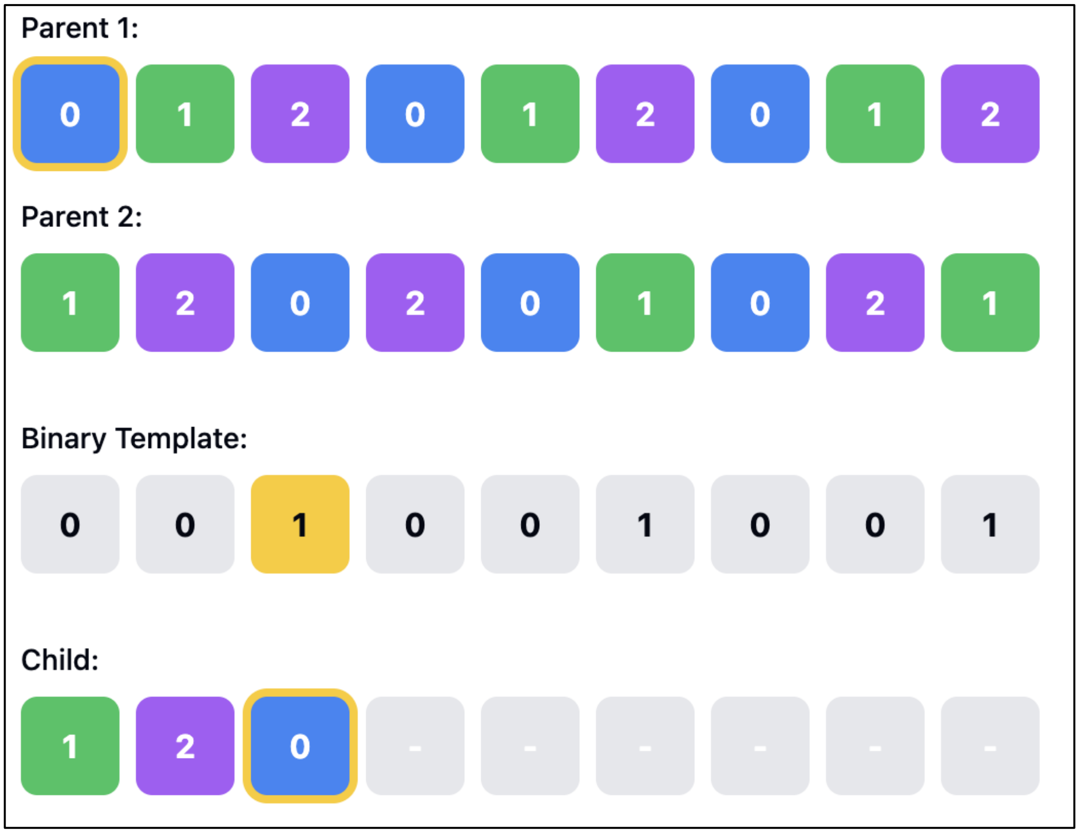
1	2	0	2	0	1	0	2	1

Binary Template:

0	0	1	0	0	1	0	0	1

Child:

-	-	-	-	-	-	-	-	-



Parent 1:

0	1	2	0	1	2	0	1	2

Parent 2:

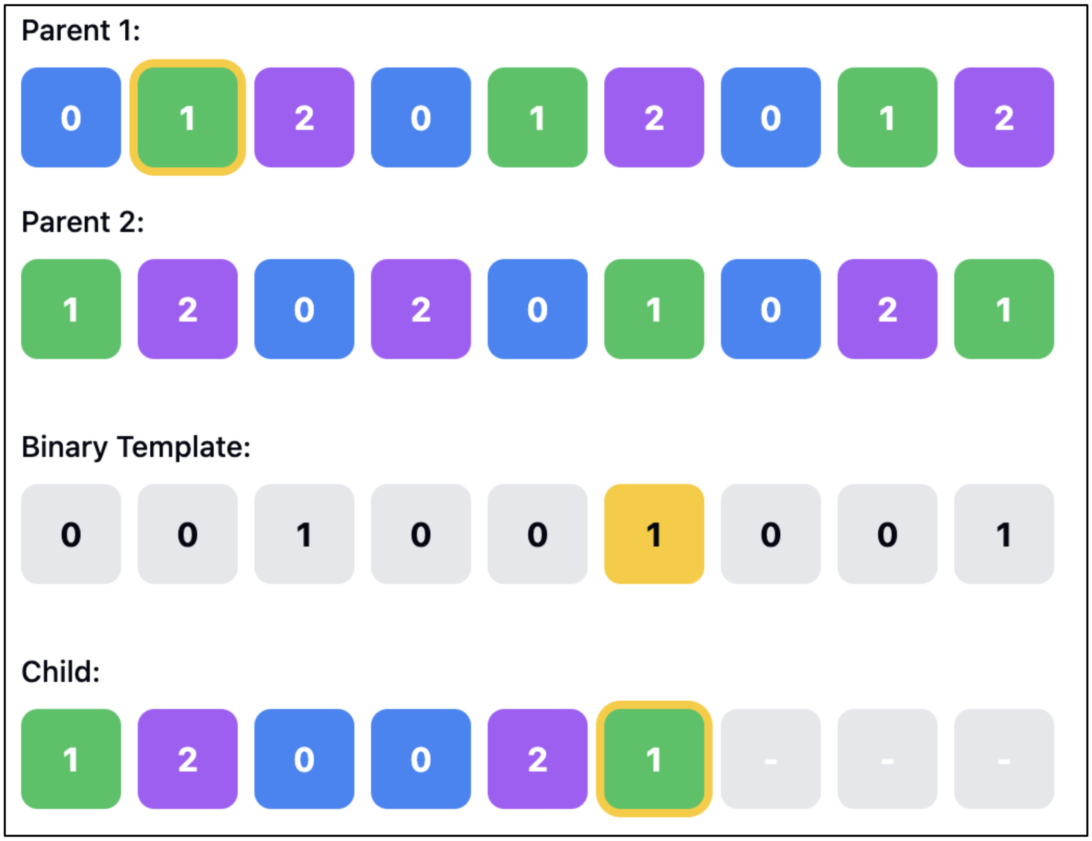
1	2	0	2	0	1	0	2	1

Binary Template:

0	0	1	0	0	1	0	0	1

Child:

1	2	0	-	-	-	-	-	-



Parent 1:

0 1 2 0 1 2 0 1 2

Parent 2:

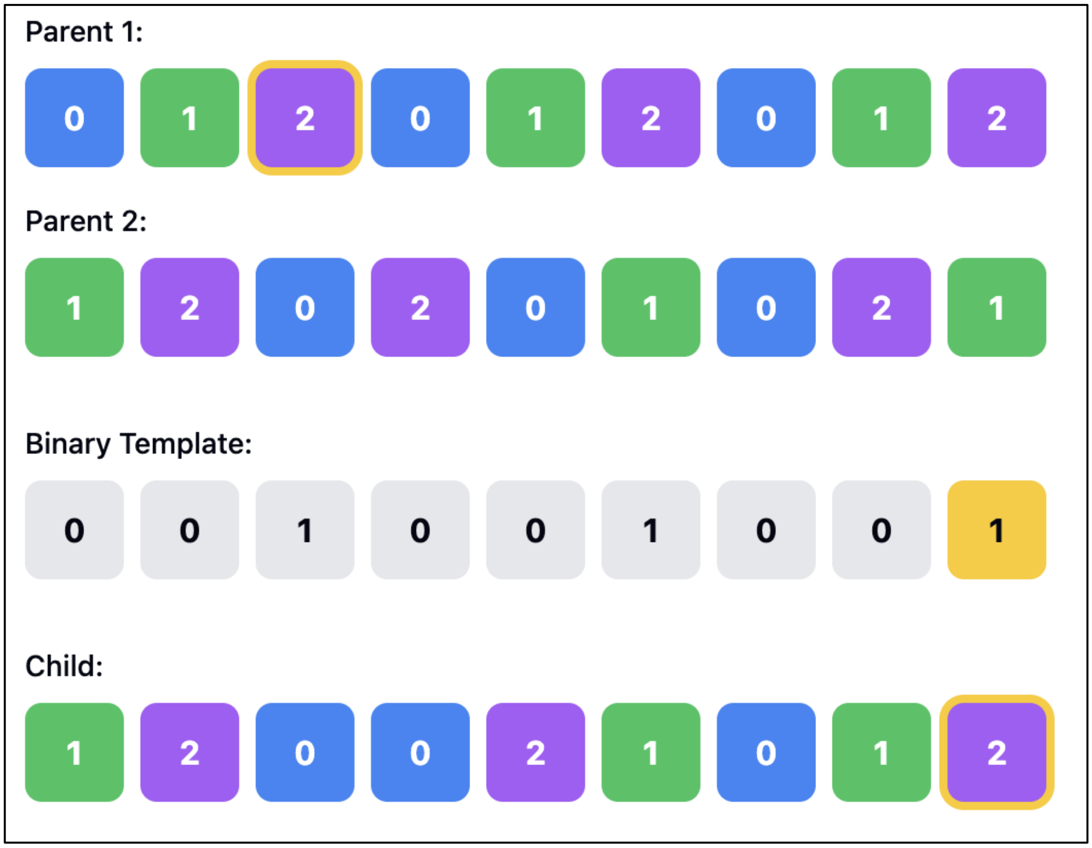
1 2 0 2 0 1 0 2 1

Binary Template:

0 0 1 0 0 1 0 0 1

Child:

1 2 0 0 2 1 - - -



Parent 1:

0 1 2 0 1 2 0 1 2

Parent 2:

1 2 0 2 0 1 0 2 1

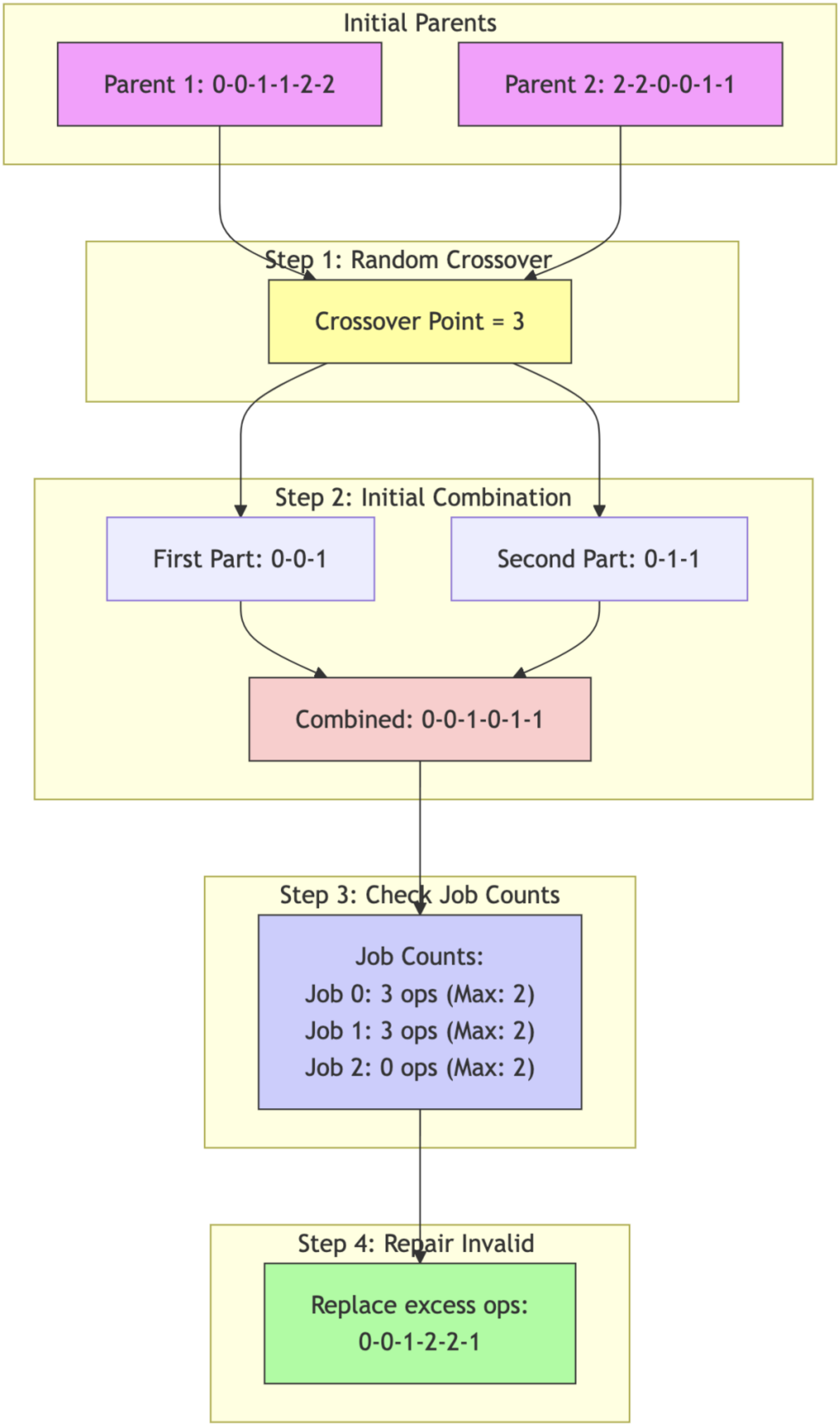
Binary Template:

0 0 1 0 0 1 0 0 1

Child:

1 2 0 0 2 1 0 1 2

2. One-Point Crossover



Population Size and Convergence Detection:

The population size is chosen based on practical considerations and theoretical guidelines:
- Default size of 100 provides a good balance between diversity and computational efficiency
- Large enough to maintain genetic diversity, out of my observations, smaller populations didn't get the optimum result but some bigger ones performed way better but took a long time to compute.
- Small enough to converge in reasonable time as the training took some time even for some medium problems, as

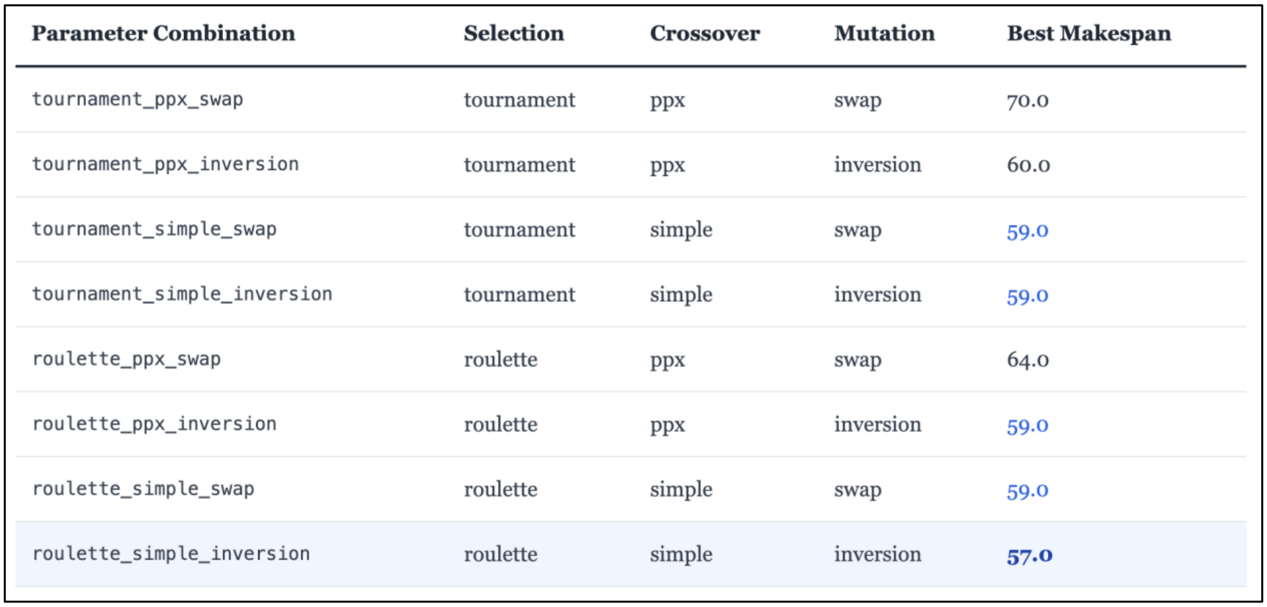
The system identifies convergence through several mechanisms:
1. Tracking population diversity using **calculate_diversity** method
2. Monitoring improvement rate in fitness values
3. Using a sophisticated convergence detection algorithm in **GAStatisticsAnalyzer**

The code uses an early stopping mechanism (MAX_STAGNANT_GENERATIONS_STOP = 30) that triggers when:
- No improvement in best fitness for 30 generations
- Population diversity falls below a threshold
- Rate of improvement becomes negligible

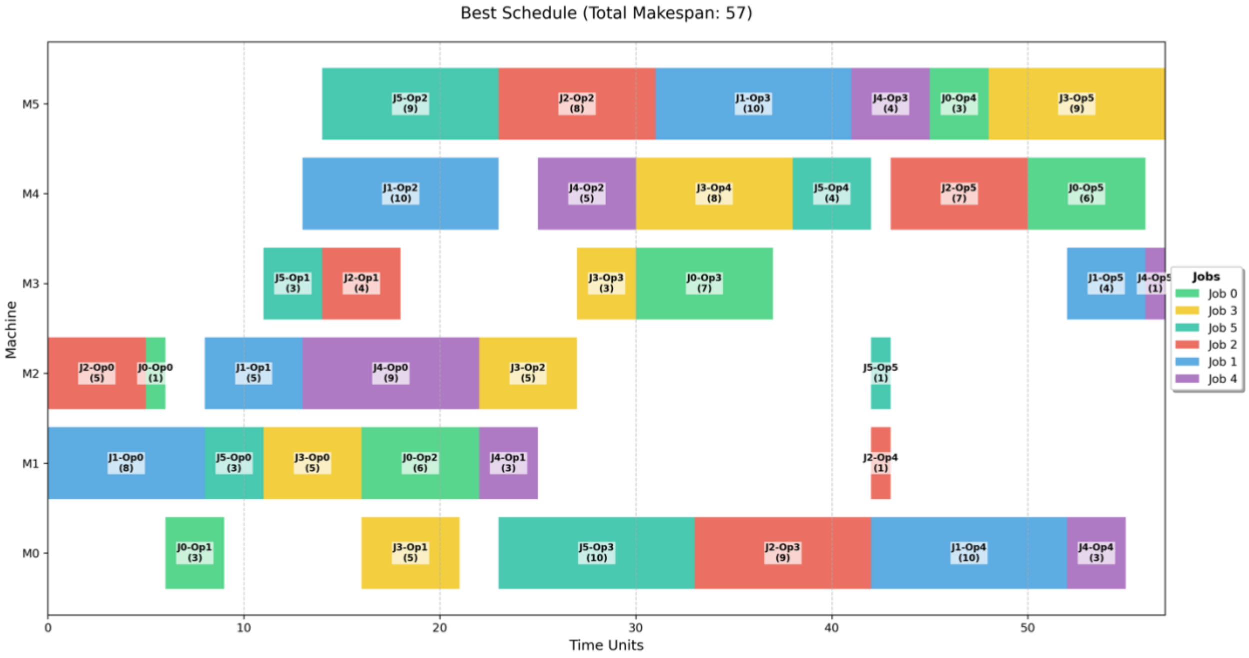
Elitism is implemented with an **elite_size** parameter (default 2), which preserves the best solutions across generations. This ensures that good solutions are not lost while still allowing for population evolution.

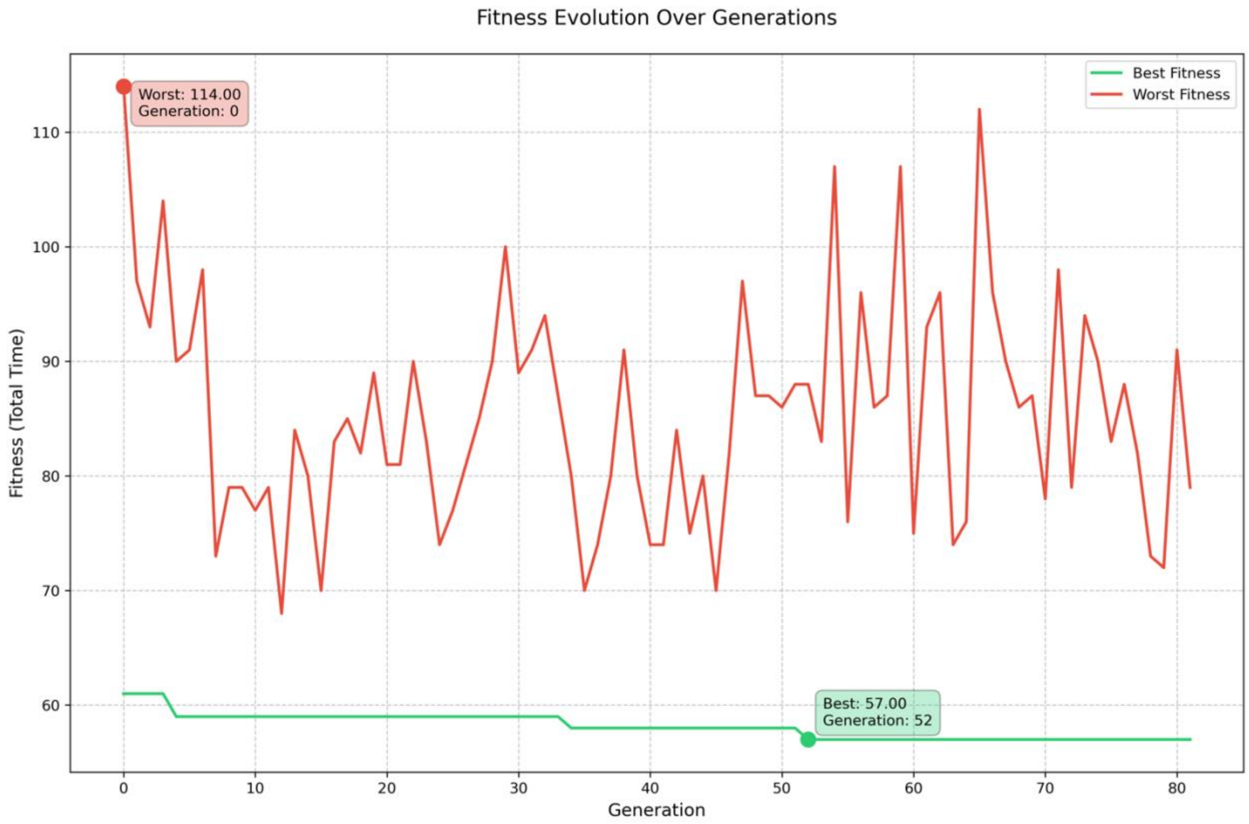
2. Results

2.1 Instance FT06 [6x6]

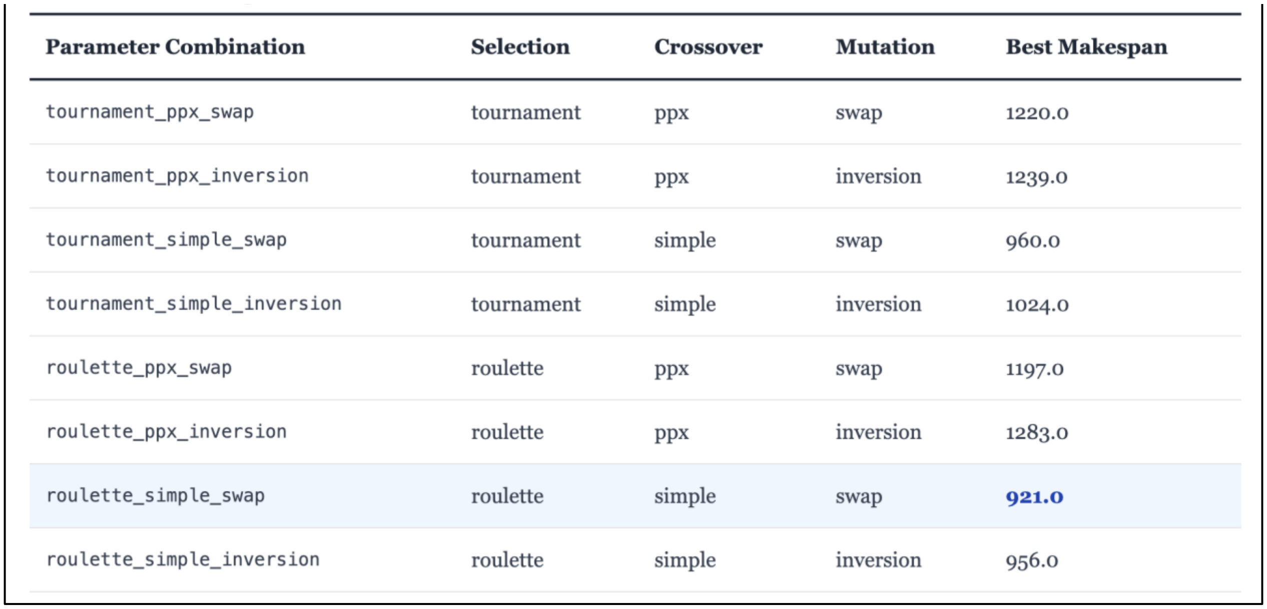


Parameter Combination	Selection	Crossover	Mutation	Best Makespan
`tournament_ppx_swap`	tournament	ppx	swap	70.0
`tournament_ppx_inversion`	tournament	ppx	inversion	60.0
`tournament_simple_swap`	tournament	simple	swap	59.0
`tournament_simple_inversion`	tournament	simple	inversion	59.0
`roulette_ppx_swap`	roulette	ppx	swap	64.0
`roulette_ppx_inversion`	roulette	ppx	inversion	59.0
`roulette_simple_swap`	roulette	simple	swap	59.0
`roulette_simple_inversion`	roulette	simple	inversion	**57.0**

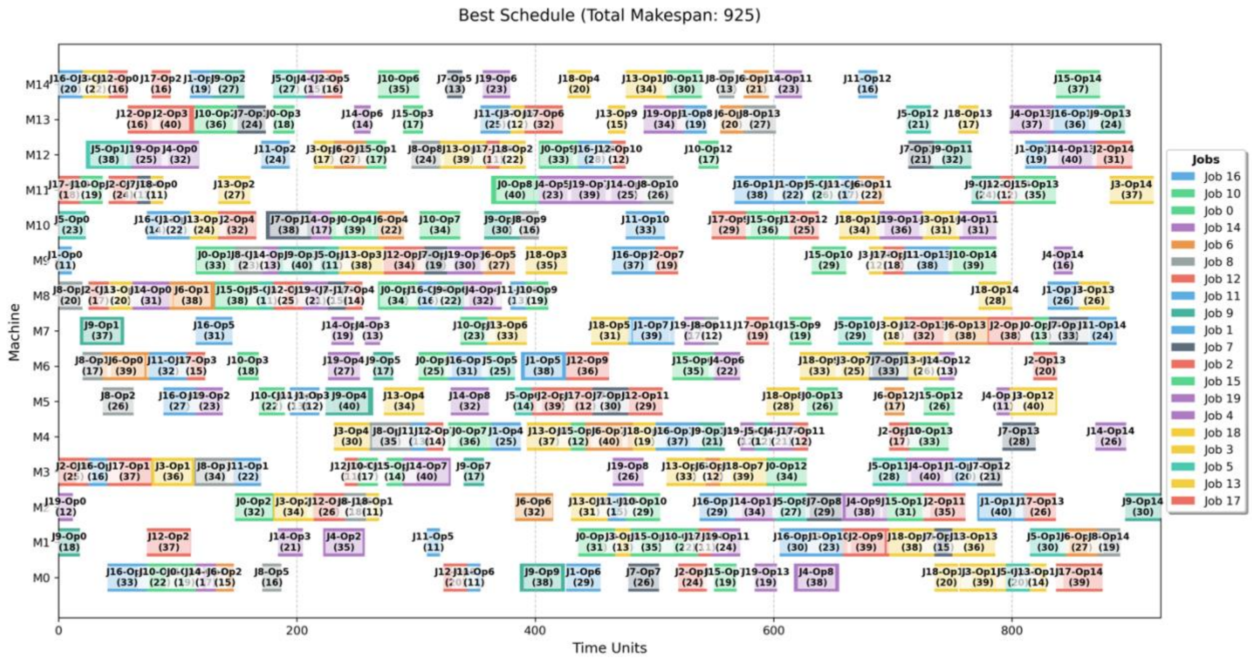


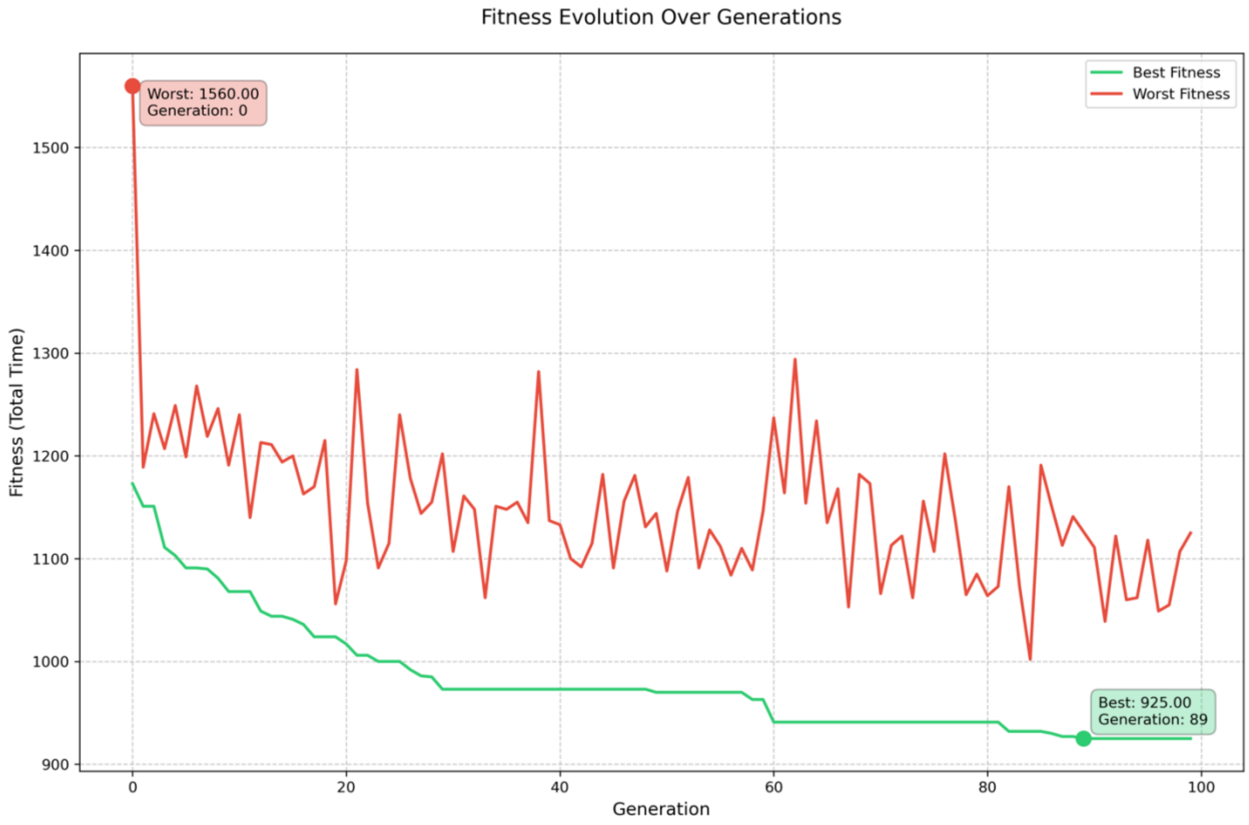


2.2 Instance ABZ8 [20x15]

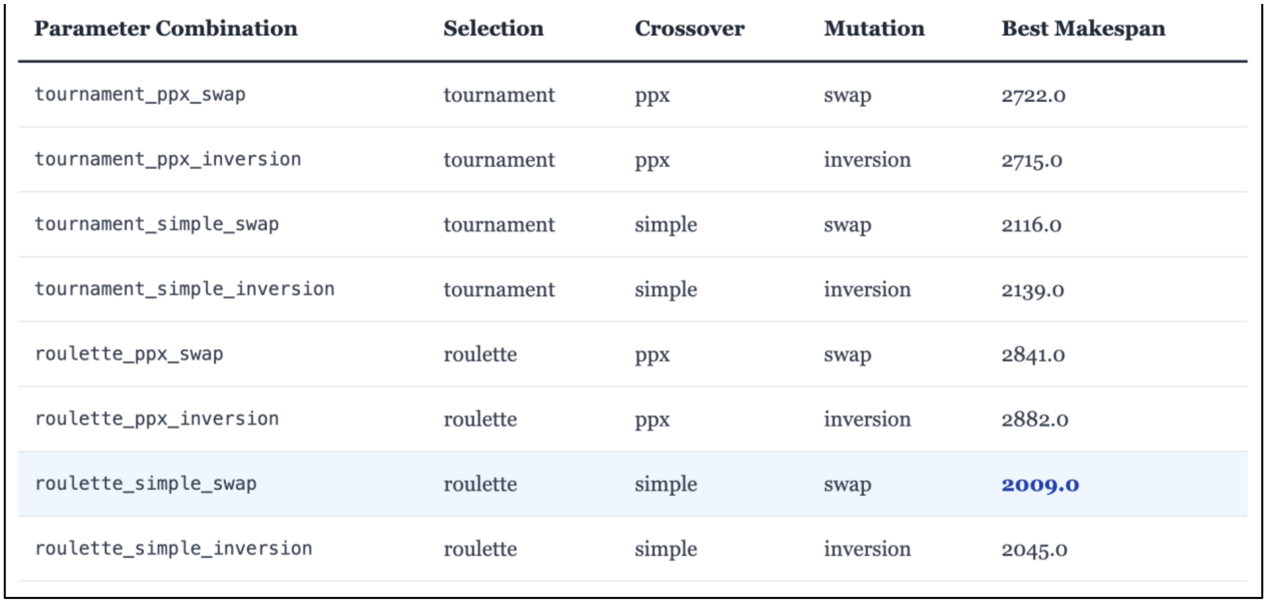


Parameter Combination	Selection	Crossover	Mutation	Best Makespan
tournament_ppx_swap	tournament	ppx	swap	1220.0
tournament_ppx_inversion	tournament	ppx	inversion	1239.0
tournament_simple_swap	tournament	simple	swap	960.0
tournament_simple_inversion	tournament	simple	inversion	1024.0
roulette_ppx_swap	roulette	ppx	swap	1197.0
roulette_ppx_inversion	roulette	ppx	inversion	1283.0
roulette_simple_swap	roulette	simple	swap	**921.0**
roulette_simple_inversion	roulette	simple	inversion	956.0

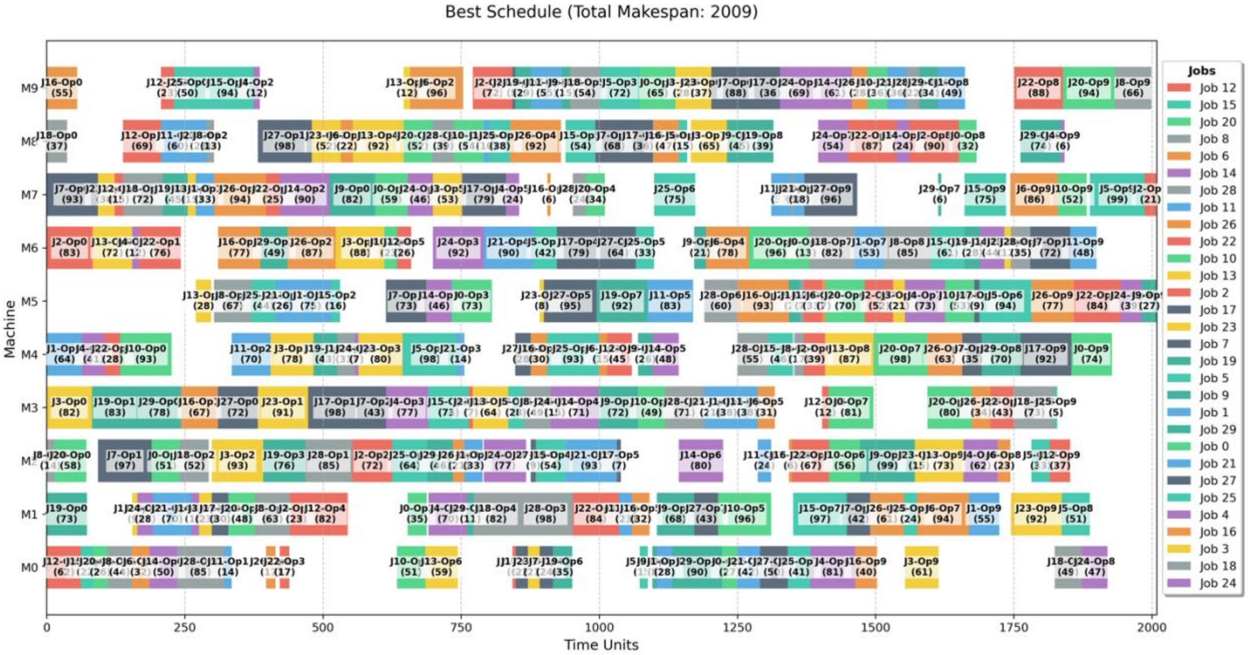

Best Schedule (Total Makespan: 925)
Machine
M14
M13
M12
M11
M10
M9
M8
M7
M6
M5
M4
M3
M2
M1
M0
Time Units
0
200
400
600
800
Jobs
Job 16
Job 10
Job 0
Job 14
Job 6
Job 8
Job 12
Job 11
Job 9
Job 1
Job 7
Job 2
Job 15
Job 19
Job 4
Job 18
Job 3
Job 5
Job 13
Job 17

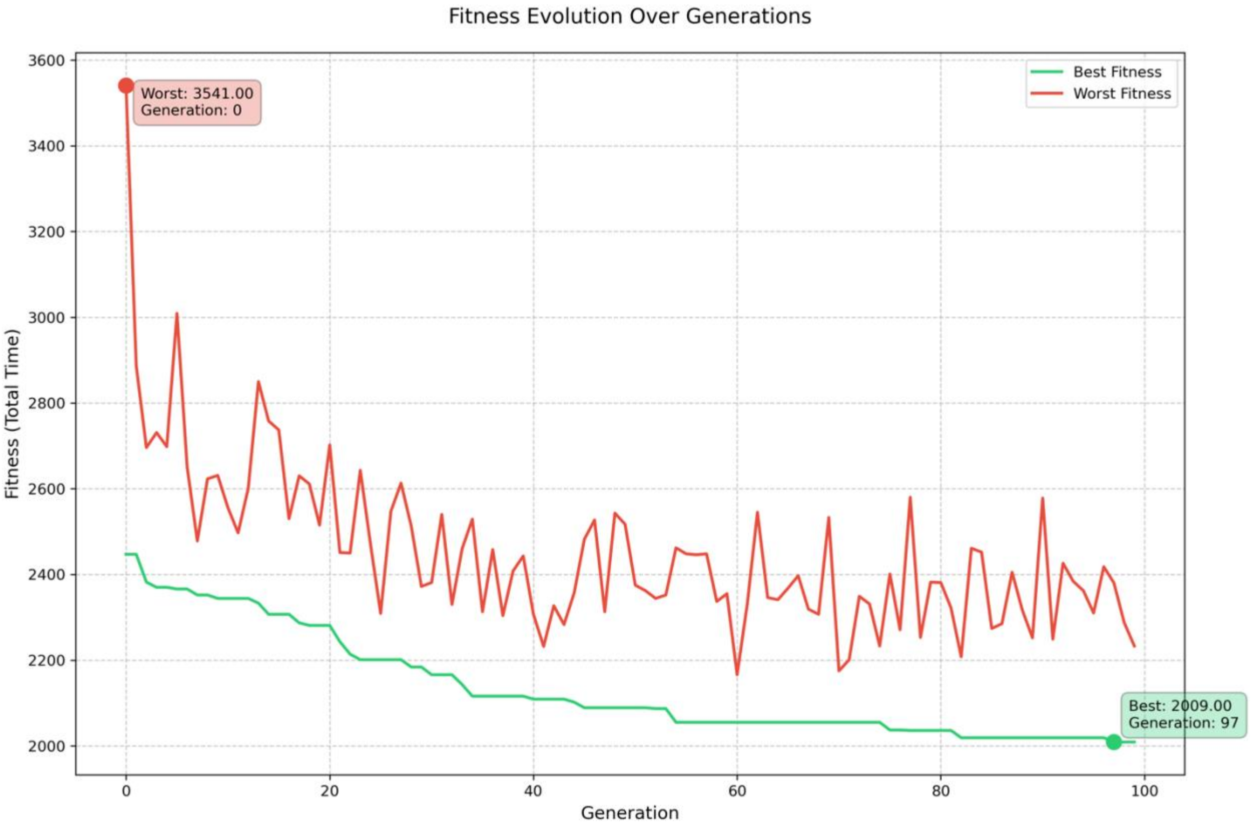

Fitness Evolution Over Generations
Fitness (Total Time)
Generation
Best Fitness
Worst Fitness
Worst: 1560.00
Generation: 0
Best: 925.00
Generation: 89
900
1000
1100
1200
1300
1400
1500
0
20
40
60
80
100

2.3 Instance LA34 [30x10]



Parameter Combination	Selection	Crossover	Mutation	Best Makespan
tournament_ppx_swap	tournament	ppx	swap	2722.0
tournament_ppx_inversion	tournament	ppx	inversion	2715.0
tournament_simple_swap	tournament	simple	swap	2116.0
tournament_simple_inversion	tournament	simple	inversion	2139.0
roulette_ppx_swap	roulette	ppx	swap	2841.0
roulette_ppx_inversion	roulette	ppx	inversion	2882.0
roulette_simple_swap	roulette	simple	swap	**2009.0**
roulette_simple_inversion	roulette	simple	inversion	2045.0




Fitness Evolution Over Generations
Best Fitness
Worst Fitness
Worst: 3541.00
Generation: 0
Best: 2009.00
Generation: 97
Fitness (Total Time)
3600
3400
3200
3000
2800
2600
2400
2200
2000
Generation
0
20
40
60
80
100