Job Shop Scheduling Problem (JSSP) Optimization Using Genetic Algorithm

---

Link to GitHub Repository

The complete code and datasets are available on the GitHub repository: https://github.com/hany019/A2-Neuoral.git

---

Objective

The primary goal of this project is to apply a Genetic Algorithm (GA) to address the Job Shop Scheduling Problem (JSSP). In addition to this, Simulated Annealing (SA) was implemented to provide a comparative analysis of the two optimization techniques. The objective is to minimize the makespan, which represents the total time required to complete all assigned jobs. This study also explores the impact of various genetic operations, including selection, crossover, and mutation, across datasets with differing complexities.

---

1. Problem Description and Dataset Overview

The Job Shop Scheduling Problem (JSSP) focuses on assigning jobs comprising sequential tasks to multiple machines under strict constraints:

- Each job must follow a predefined sequence of tasks.
- Machines can handle only one task at a time.
- Tasks, once initiated, must run uninterrupted until completion.

The datasets for this project were sourced from the OR-Library. The datasets used are as follows:

- **abz5 (10x10)**: Contains 10 jobs and 10 machines.
- **abz7 (15x20)**: Contains 20 jobs and 15 machines.
- **abz9 (15x20)**: Contains 20 jobs and 15 machines.
-

Dataset Description

- Each instance begins with a description line, followed by the number of jobs and machines, and then lines for each job, listing the machine number and processing time for each step.
- URL: https://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/jobshop1.txt

---

2. Chromosome Representation and Adaptations

Chromosome Representation

Each chromosome represents a possible task schedule for the JSSP. Each gene inside a chromosome encodes a task, defined by its job ID, machine, and processing duration. The sequence guarantees that precedence constraints are adhered to.

Example:

- A chromosome for a 3-job, 3-machine problem:
  - JOB 0 = [(0, 3), (1, 2), (2, 2)]
  - JOB 1 = [(0, 2), (2, 1), (1, 4)]
  - JOB 2 = [(1, 4), (2, 3)]
- This representation ensures all tasks are included and respects job dependencies.

Selection Method

- **Tournament Selection**: was employed to balance exploitation and exploration. A subset of chromosomes is randomly chosen, and the fittest among them is selected.

Crossover Methods

1. **Two-Point Crossover**: Two random points in the parent chromosomes are selected, and the segment between them is swapped.
2. **Uniform Crossover**: Each gene in the offspring is randomly inherited from one of the parents.

Mutation Methods

1. **Always One Mutation Per Chromosome**: A single gene in the chromosome is randomly selected, and its position is swapped with another random gene.
2. **Independent Gene Mutation**: Each gene in the chromosome is mutated independently with a fixed probability (mutation rate). The mutation involves swapping the gene's position with another random gene.

Both mutation methods align with techniques discussed in the course slides.

Elitism

Elitism ensures that the best-performing chromosomes from the current generation are directly carried over to the next generation. This guarantees that the best solutions are not lost during the evolution process. For this implementation, the number of elite chromosomes retained (ga_elite_count) is set to 2.

Population parameters

- **Population Size:** 50, based on initial testing and literature.
- **Convergence:** The system is considered to have converged when there is no significant improvement in the best fitness value over successive generations.

---

3. Validation of Chromosomes

Chromosomes are validated to ensure they represent feasible solutions by:

- Ensuring all tasks for each job are present and appear in the correct order.
- Verifying machine availability: Each machine processes only one task at a time.
- Adhering to job constraints: No task starts until the previous task for the same job is completed.

---

4. Fitness Function

The fitness function calculates the makespan by:

- Initializing completion times for all jobs and machines.
- Determining each task's start time based on machine availability and previous task completion.
- Update the completion time for the machine and job.
- Outputting the maximum completion time across all machines as the fitness value.

---

5. Results

Dataset: abz5 (10x10)

- **Parameter Combinations:**

```python
# Define GA parameter combinations
ga_parameter_combinations = [
    {"population_size": 50, "mutation_rate": 0.1, "crossover_rate": 0.8, "elitism": 2},
    {"population_size": 100, "mutation_rate": 0.05, "crossover_rate": 0.7, "elitism": 2},
    {"population_size": 50, "mutation_rate": 0.2, "crossover_rate": 0.6, "elitism": 1},
    {"population_size": 100, "mutation_rate": 0.15, "crossover_rate": 0.9, "elitism": 3},
    {"population_size": 75, "mutation_rate": 0.1, "crossover_rate": 0.8, "elitism": 2},
    {"population_size": 50, "mutation_rate": 0.05, "crossover_rate": 0.9, "elitism": 1},
]
```

Technique	Best Makespan	Best Chromosome
Genetic Algorithm (GA)	134 ,164 , 308 , 268 ,293 ,459	[(1, (3, 50)), (1, (3, 50)), (9, (5, 59)), (4, (6, 95)), (3, (8, 67)),…….]
Simulated Annealing (SA)	1171,1266,1126, 1228,1233,1159	[(9, (6, 58)), (2, (0, 83)), (1, (2, 94)), (5, (8, 80)),, …]

- **Fitness Evolution:** The GA rapidly converged to a better solution compared to SA, as shown in the fitness evolution plot.

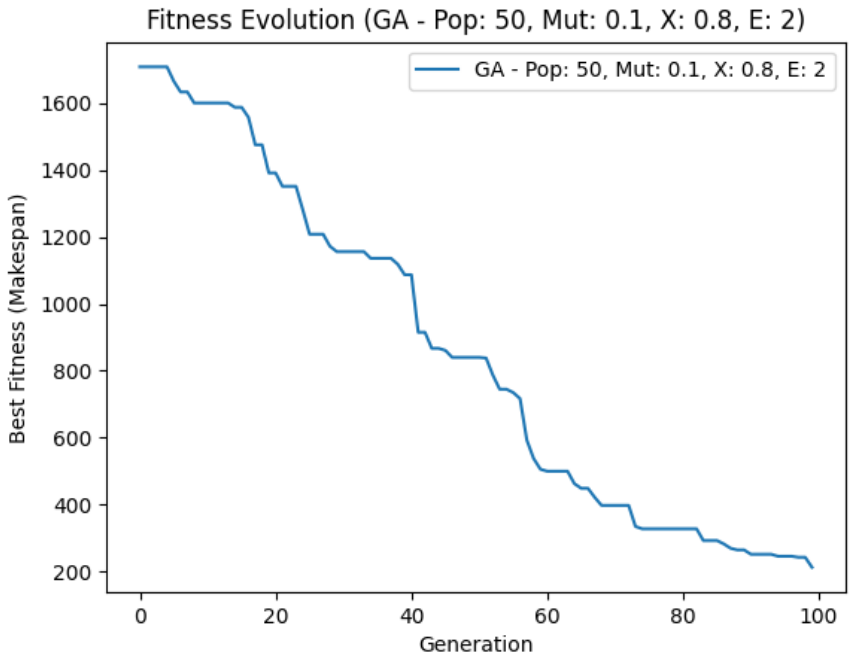

Figure 1A: Fitness evolution for GA on the abz5 dataset(Makespan 134).

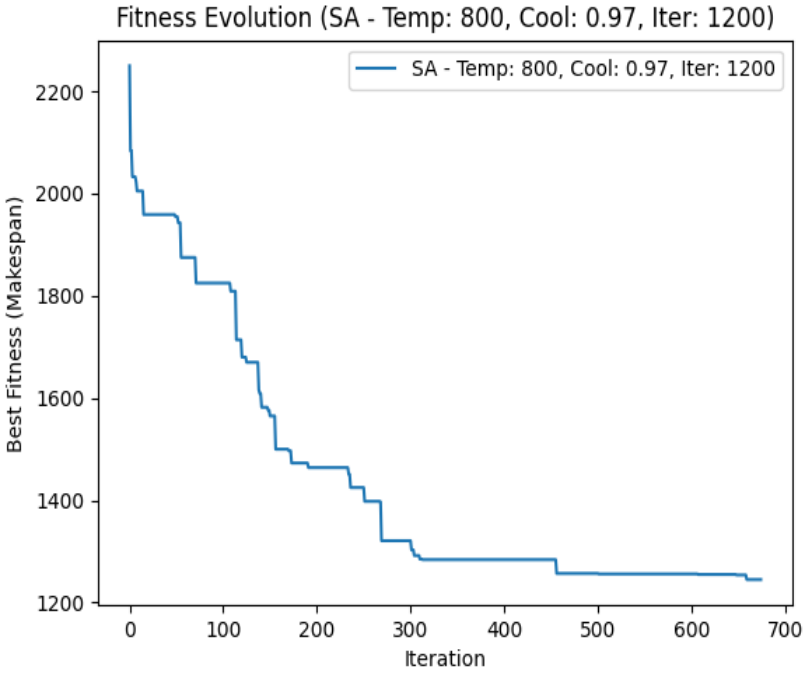

Figure 1B: Fitness evolution for SA on the abz5 dataset (Makespan 1126).

.

Dataset: abz7 (15x20)

Technique	Best Makespan	Best Chromosome
Genetic Algorithm (GA)	223 , 202, 87, 377 , 285, 359	[(15, (0, 13)), (9, (4, 15)), (12, (11, 12)), (2, (1, 14)), …]
Simulated Annealing (SA)	780 , 870, 737 , 832 , 808, 843	[(16, (3, 25)), (7, (14, 39)), (3, (4, 30)), (7, (13, 39)), …]

- **Fitness Evolution:** Similar to abz5, the GA significantly outperformed SA for this dataset.

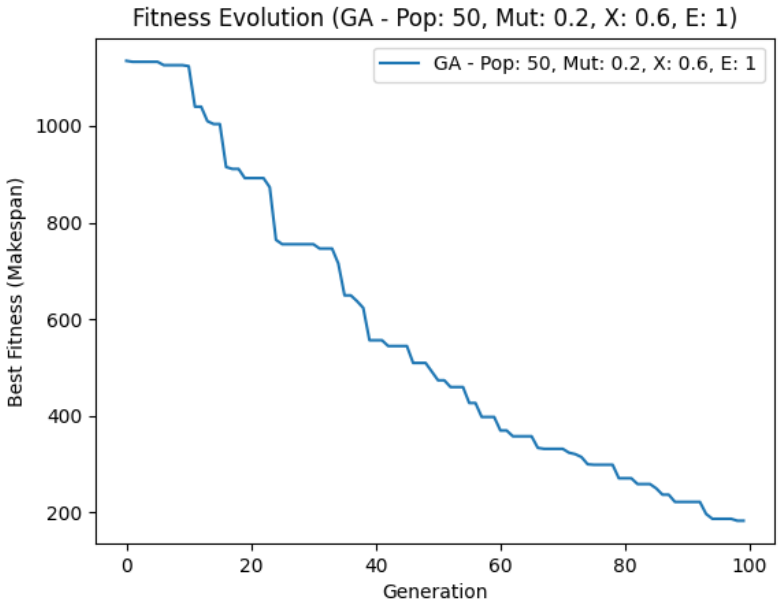

Figure 2A: Fitness evolution for GA on the abz7 dataset(Makespan 87)

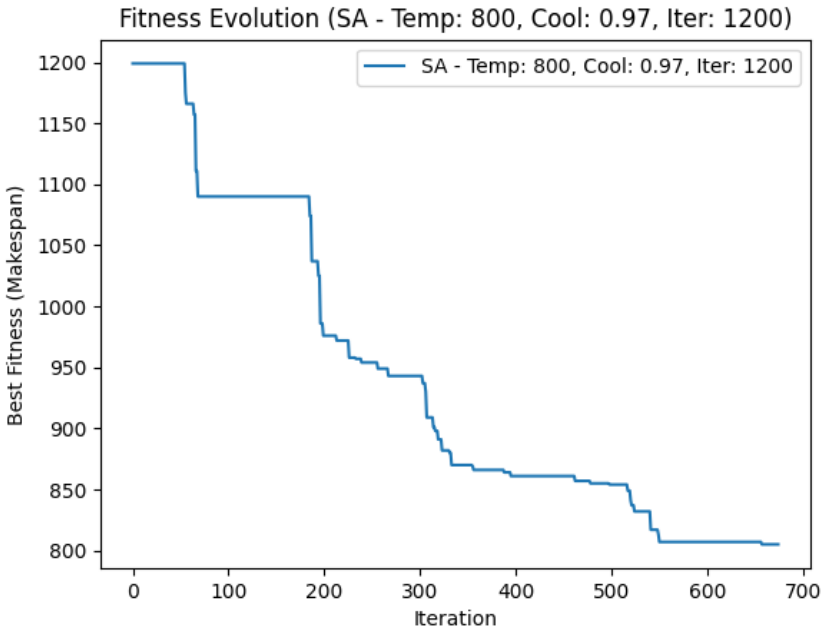

Figure 2B: Fitness evolution for SA on the abz7 dataset(Makespan 737)

Dataset: abz9 (15x20)

Technique	Best Makespan	Best Chromosome
Genetic Algorithm (GA)	112 , 309, 268, 406 , 172, 298	[(15, (0, 13)), (9, (4, 15)), (12, (11, 12)), (2, (1, 14)), …]
Simulated Annealing (SA)	776 , 909, 774 , 805 , 849, 912	[(16, (3, 25)), (7, (14, 39)), (3, (4, 30)), (7, (13, 39)), …]

- **Fitness Evolution:** Similar to previous datasets, GA significantly outperformed SA.

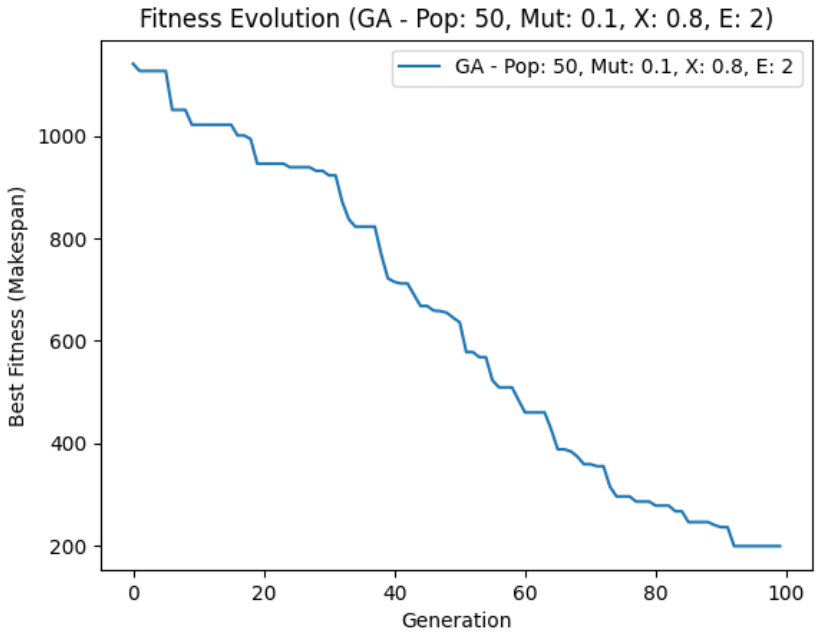

Figure 3A: Fitness evolution for GA on the abz9 dataset(Makespan 112)

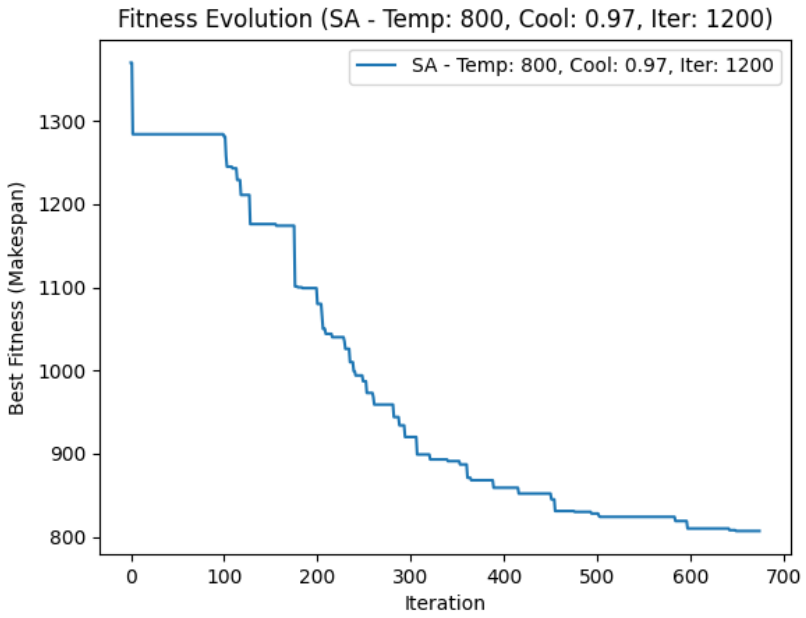

Figure 3B: Fitness evolution for SA on the abz9 dataset(Makespan 774)

---

6. Discussion and Comparison

Performance Analysis

1. **Genetic Algorithm (GA):**
   - Achieved superior performance across all datasets, producing significantly lower makespan values.
   - Efficient exploration and exploitation were achieved using elitism and diverse crossover/mutation strategies.
2. **Simulated Annealing (SA):**
   - Performed reasonably well but was unable to match the results of the GA.
   - SA's performance highly depends on parameter tuning (initial temperature and cooling rate).
3. **Scalability:**
   - GA demonstrated robustness and scalability, effectively handling larger datasets.

- SA's performance degraded with increased problem complexity, underscoring its limitations.

---

7. Conclusion

The Genetic Algorithm (GA) has demonstrated its effectiveness in solving the Job Shop Scheduling Problem, outperforming Simulated Annealing (SA) in all tested scenarios. The success of the GA can be attributed to its robust genetic operations and the incorporation of elitism. Future research could explore hybridizing GA with SA to leverage their respective strengths or further refine the parameter settings to enhance scalability.