Jangeun Kim 20236471

Term Project Final Presentation

[Resource-Constrained Project Scheduling Problem, RCPSP]

Contents

- 1. What is the RCPSP?
- 2. Problem Definition
- 3. GA design
- 4. Performance result
- 5. Conclusion & future plan

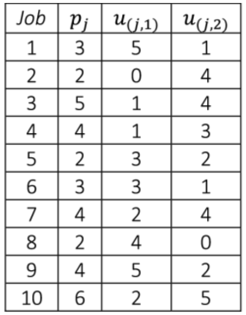
1. What is the RCPSP?

1. What is the RCPSP?

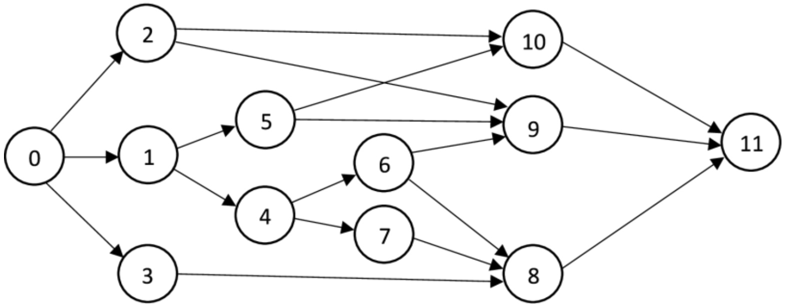
- Definition

✓ **Resource-Constrained Project Scheduling Problem (RCPSP)** is a combinatorial optimization problem in the field of operations research and project management.

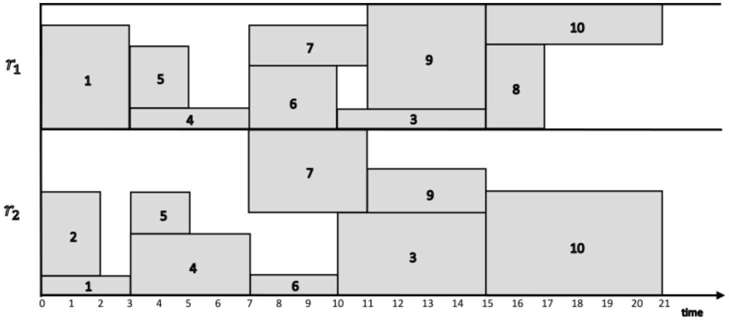
✓ It involves scheduling a set of activities or tasks with given durations and resource requirements in such a way that the project is completed as quickly as possible while respecting resource constraints.



Job	p_j	$u_{(j,1)}$	$u_{(j,2)}$
1	3	5	1
2	2	0	4
3	5	1	4
4	4	1	3
5	2	3	2
6	3	3	1
7	4	2	4
8	2	4	0
9	4	5	2
10	6	2	5



AoN network for project instance



optimal schedule for instance

1. What is the RCPSP?

- Key considerations (Constraints)

✓ Activities

These are tasks or jobs that need to be scheduled.

☞ Each activity has a defined duration, a set of required resources, and precedence relationships with other activities.

✓ Resources

There are limited resources available for executing the activities.

☞ Resources can include labor, machinery, materials, or any other constraints that can affect the scheduling

✓ Precedence Relationships

These dependencies are represented as precedence relationships.

☞ Activities may have dependencies on each other, meaning that certain activities must be completed before others can start.

1. What is the RCPSP?

- Main objective

✓ The objective in RCPSP is to find a schedule that **minimizes the project's makespan** while satisfying the resource constraints and respecting the precedence relationships between activities

✓ RCPSP is known to be an **NP-hard problem**, which means that finding an optimal solution can be computationally challenging, especially for large and complex projects.

$$\text{Minimize} \quad \sum_{t=0}^{T} t \cdot x_{J+1,t}$$

subject to

$$\sum_{t=0}^{T} x_{jt} = 1 \qquad j = 0, \ldots, J+1$$

$$\sum_{t=0}^{T} t \cdot x_{ht} \le \sum_{t=0}^{T} (t - p_j) \cdot x_{jt} \qquad j = 0, \ldots, J+1, \quad h \in P_j$$

$$\sum_{j=1}^{J} \sum_{q=t}^{t+p_j-1} r_{j,k,t+p_j-q} \cdot x_{jq} \le R_{kt} \qquad k = 1, \ldots, K, \quad t = 1, \ldots, T$$

$$x_{jt} \in \{0, 1\} \qquad j = 0, \ldots, J+1, \quad t = 0, \ldots, T$$

1. What is the RCPSP?

- What is the difference between RCPSP and JSSP?

Job Shop Scheduling Problem (JSSP)

✓ <u>Scope</u>

- **RCPSP** is primarily concerned with **scheduling activities in a project environment**. The activities are tasks that need to be executed to complete a project.
- **JSSP** is focused on **scheduling jobs in a manufacturing environment**, particularly in job shops where different jobs require different sequences of operations on various machines.

✓ <u>Constraints</u>

- **RCPSP** : Activities, Resources, Precedence
- **JSSP** : Jobs and Operations , Machines, Precedence

✓ <u>Objective</u>

- **RCPSP/JSSP** :
  minimize the makespan

✓ <u>Applications</u>

- **RCPSP** is commonly used in **project management scenarios**, such as **construction projects**, **software development projects**, and manufacturing projects where tasks are interdependent.
- **JSSP** is commonly used in **manufacturing settings**, such as **job shops**, where different types of jobs with varying processing requirements need to be scheduled on **available machines.**

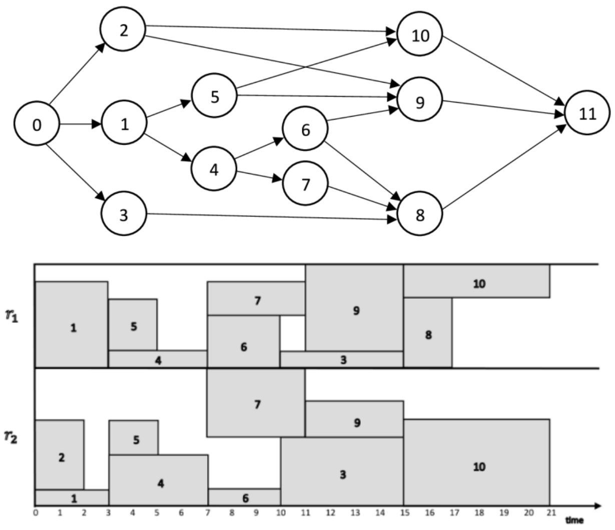
1. What is the RCPSP?

- What is the difference between RCPSP and JSSP?

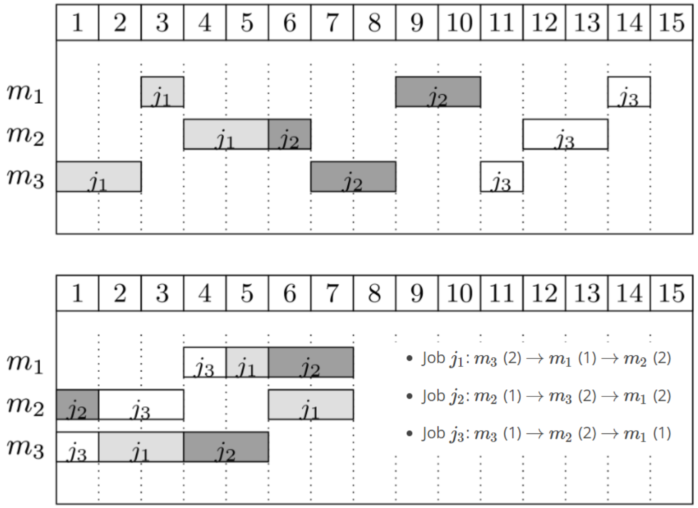
✓ In summary,

- **RCPSP** is more aligned with **project-based scheduling**, where tasks have precedence relationships, and the goal is to optimize the overall project completion time.

- **JSSP** is focused on **manufacturing environments**, where jobs involve multiple operations on different machines, and the objective is often to minimize makespan.



RCPSP


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
m_1
m_2
m_3
j_1
j_2
j_3
Job j_1: m_3 (2) → m_1 (1) → m_2 (2)
Job j_2: m_2 (1) → m_3 (2) → m_1 (2)
Job j_3: m_3 (1) → m_2 (2) → m_1 (1)

JSSP

1. What is the RCPSP?

- Reasons for using GA to solve RCPSP

  ✓ Applying Genetic Algorithms (GAs) to the resource-constrained project scheduling problem (RCPSP) is advantageous for the following key reasons :

Diverse Exploration

Parallel Processing

Robustness to Constraints

Automated Search and Adaptability

➢ **In summary,**
GAs are favored in RCPSP due to their ability to explore diverse solutions, handle complex constraints, perform parallel processing, and adapt to various scheduling scenarios, which gives them a competitive edge over other heuristic methods.

2. Problem Definition

2. Problem Definition

- Problem Description

```
[ 0,  7, 10,  7,  7,  2,  8,  6,  1,  7, 10,  4,  5,  4,  6,  5,  1,
  5,  7,  6,  6, 10,  9,  2,  4,  5,  2, 10,  3,  6,  9,  1,  5,  2,
  8,  5,  6,  6,  1, 10,  3,  2,  7,  2,  1,  1,  3,  6,  1,  6,  4,
  9, 10,  5,  3,  3,  4,  3, 10,  5,  9,  3,  3,  9,  2,  8,  2,  7,
  1,  4,  8,  9,  5,  2,  5,  3,  4,  2,  9,  8,  7, 10,  4,  2,  2,
  7,  6,  7,  9,  9,  5,  0]
```

[Duration for each activity]

✓ **Project / Activities / Sum of duration** : 1 / 92 (including dummy) / 477 days

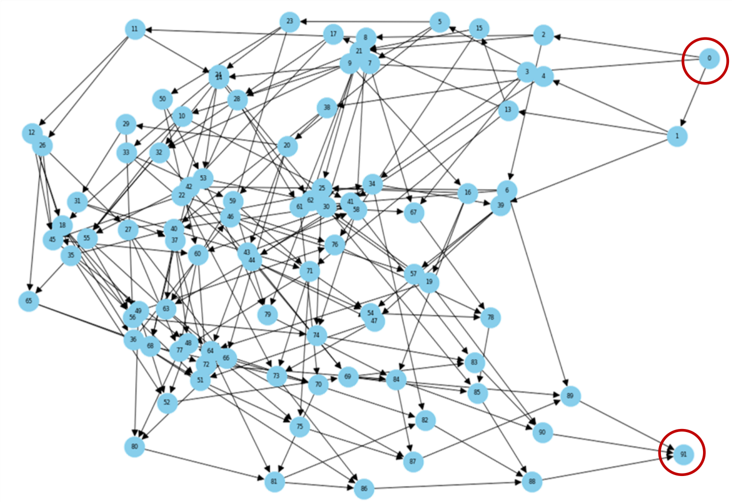
[Resource requirement for each activity]

✓ **Resources type / total amount** : 4 / [18, 21, 20, 18]

```
[[ 0,  0,  0,  0],
 [ 0,  0,  5,  6],
 [ 1,  1,  8,  0],
 [10,  9,  2,  8],
 [10,  6,  1,  3],
 [ 4,  9,  1,  1],
 [ 8,  7,  0,  0],
 [ 9,  7,  7,  4],
 [ 1,  3,  0,  2],
 [ 3,  3,  6,  0],
 [ 6,  4,  5,  0],
 [ 8,  5,  1,  0],
 [ 2,  4,  0,  6],
 [ 0,  6,  4,  0],
 [ 0,  3, 10,  9],
 [ 1,  0,  1,  5],
 [ 5,  5,  3,  2],
 [10,  0,  0,  3],
 [ 4,  0,  6,  7],
 [ 4,  3,  6,  0],
 [ 1,  5,  0,  1],
 [ 7,  2,  1,  1],
 [ 0,  0,  4,  8],
 [ 4,  3,  3,  5],
 [ 0, 10,  0,  2],
 [ 5,  5,  5,  0],
 [10,  9,  0,  0],
 [ 9,  0,  2,  7],
 [ 9,  0,  9,  1],
 [ 4, 10,  0,  0],
 [ 0,  7,  2,  2],
 [ 0,  0,  2,  1],
 [ 8,  2,  0,  8],
 [ 8, 10,  5,  2],
 [ 9,  6,  9,  0],
 [ 0,  0,  6,  6],
 [ 6,  2,  0,  1],
 [ 0,  6,  2, 10],
 [10,  4, 10,  0],
 [ 0,  7,  1,  3],
 [ 7,  0,  2,  2],
 [ 0,  7, 10,  3],
 [ 4,  0,  4,  4],
 [ 0,  8,  9,  7],
 [ 0, 10,  0,  0],
 [ 6,  7,  3,  1],
```

```
[ 4,  8,  1,  0],
[ 8,  0,  0,  8],
[ 2,  3,  0,  5],
[10,  6, 10,  2],
[ 0,  0,  7,  9],
[ 9,  8,  8,  9],
[ 1,  3,  1,  8],
[ 8,  0, 10, 10],
[ 8,  0,  7,  8],
[ 4,  5,  8,  1],
[ 8,  4,  6,  7],
[ 0,  7,  6,  0],
[ 9,  7,  8, 10],
[ 3,  8,  0,  0],
[ 2, 10,  4,  2],
[ 6,  8,  5,  0],
[ 7,  0,  1,  4],
[ 3,  3,  8,  0],
[ 0,  8,  3,  0],
[ 5, 10,  9,  0],
[ 5,  1,  4,  3],
[ 2,  8,  0,  4],
[ 4,  6,  4,  3],
[ 0,  6,  1,  2],
[ 0,  3,  6,  7],
[ 0,  7,  8,  4],
[ 0,  3, 10,  0],
[ 0,  7,  0,  9],
[ 0,  0,  4,  7],
[10,  5,  8,  8],
[ 0,  6,  4,  1],
[ 0,  9,  8,  7],
[10,  0,  9,  4],
[ 0,  3,  4,  6],
[10,  6,  7,  0],
[ 2,  5,  5,  8],
[ 0,  0,  7, 10],
[ 2,  6,  0,  1],
[ 2,  3,  0,  4],
[ 9,  9,  6,  0],
[ 1,  1,  8,  4],
[ 0, 10,  5,  5],
[10,  9,  8,  6],
[ 4,  2,  0,  5],
[ 0,  2,  1,  4],
[ 0,  0,  0,  0]]
```

✓ **Precedence Relationships**



```
[[],
 [0],
 [0],
 [0],
 [1],
 [4],
 [2],
 [5],
 [2],
 [3],
 [9],
 [8],
 [11],
 [1],
 [7, 11],
 [13],
 [9],
 [13],
 [10, 12],
 [16],
 [5],
 [2, 15],
 [14],
 [5],
 [88, 89, 90]]
```

```
[23],
[6, 16, 21],
[11],
[26],
[17, 21],
[20],
[7, 19, 24],
[29],
[9, 14, 28],
[24],
[3, 27],
[12, 32],
[18],
[28, 31],
[4],
[1, 34],
[30, 35, 38],
[4, 10],
[17, 25],
[20],
[32, 33, 34],
[12, 22, 26],
[23, 36],
[39, 44],
[37, 45],
```

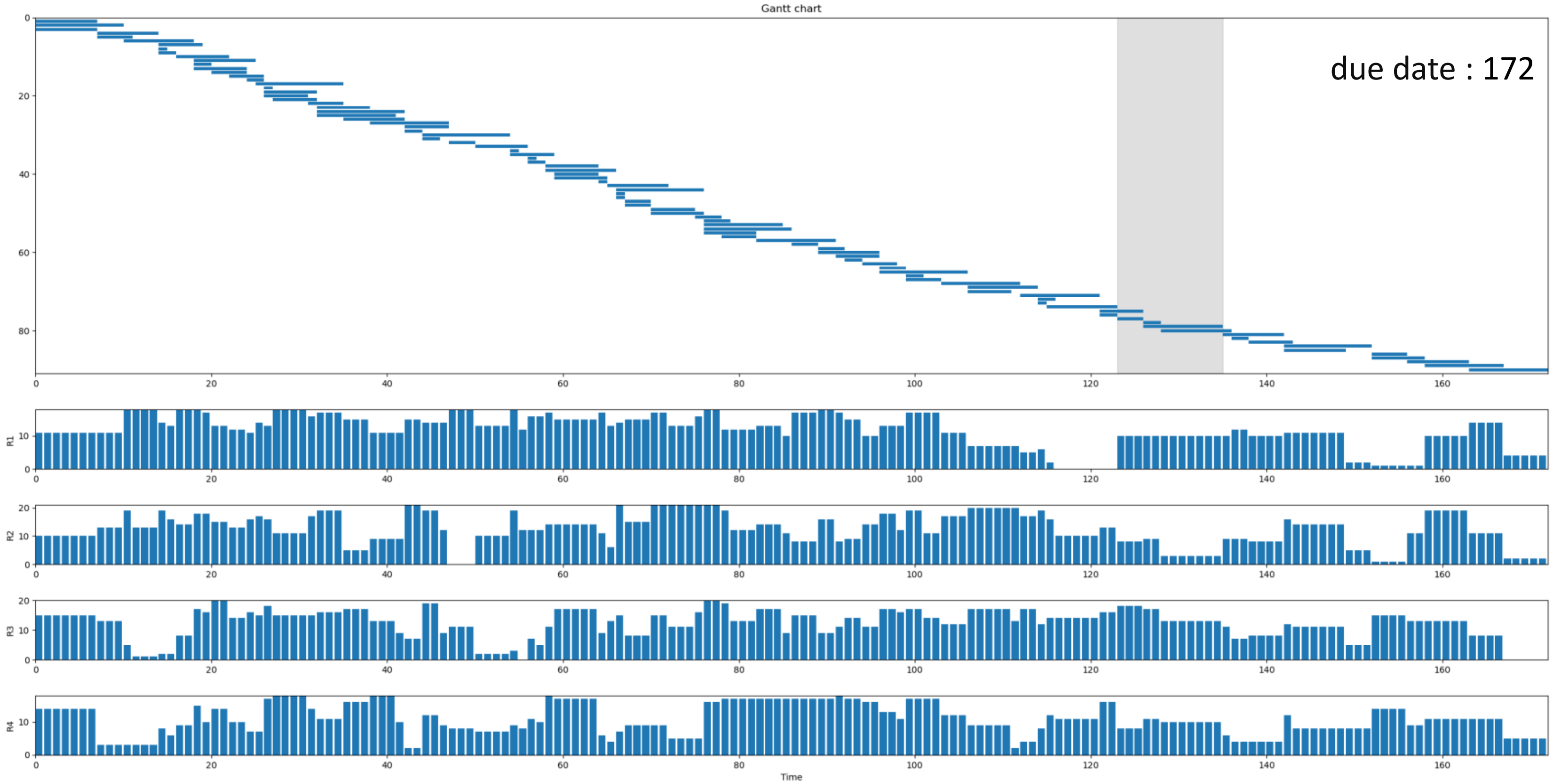
```
[18],
[23],
[35, 36, 47],
[35, 48, 49],
[24, 50],
[39, 44, 46],
[10, 32, 53],
[18, 29, 55],
[39, 46],
[8, 43, 53],
[33, 38],
[41, 45, 50],
[6, 8, 28],
[14, 21, 57],
[40, 41, 59],
[19, 43, 55],
[26, 55],
[20, 27, 63],
[3, 7, 61],
[37, 42, 65],
[43, 44, 52],
[61, 66, 68],
[13, 31, 59],
[37, 60, 65],
[22, 47, 71],
```

```
[25, 56, 58],
[51, 64, 71],
[15, 22, 49],
[27, 60, 71],
[54, 67, 76],
[53, 59, 62],
[49, 66, 77],
[64, 70, 80],
[34, 48, 81],
[41, 69, 74],
[16, 72, 79],
[30, 69, 78],
[36, 73, 81],
[66, 75, 84],
[82, 85, 86],
[6, 69, 87],
[19, 83, 84],
```

[Precedence relationships for each activity]

2. Problem Definition

- How to solve this problem?
  - ✓ Example of feasible activity string



due date : 172

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,
37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,   68, 69, 70, 71,
72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91]

3. GA design

3. GA design

- GA design for RCPSP

✓ **Representation of a chromosome :** Order type based on Integer

e.g. [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91]

✓ **Fitness function :** Minimize the makespan (=objective function Minimize $\sum_{t=0}^{T} t \cdot x_{J+1,t}$)

✓ **Generation of population and termination criteria**

- **Initialization population :** making feasible activity Integer string considering constraints

☞ Step1. the dummy source activity is started at time 0.

☞ Step2. Sorting based on the Latest Finish Time

☞ Step3. Identification and selection of activities that satisfy the precedence condition.

* If there are multiple activities that satisfy multiple Precedence conditions, randomly select one activity from the pool.

☞ Step4. Through repeating the above steps, generate various strings that satisfy the constraints.

- **Termination Criteria :** a number of generations(<=300)

*** Detailed content is explained in Chapter 4.**

3. GA design

• GA design for RCPSP

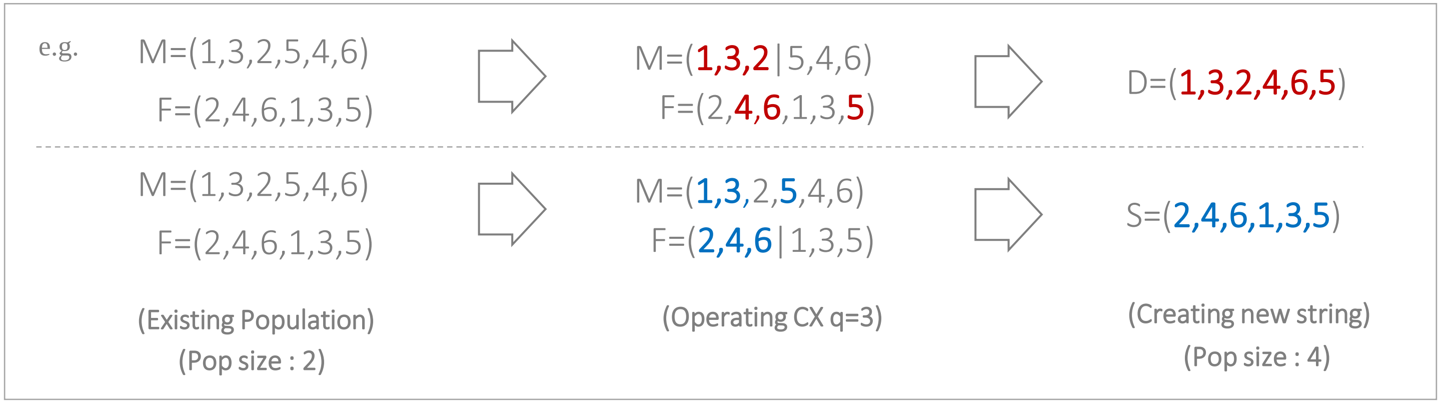
✓ Operators

- **Crossover :** One-Point Crossover

  ☞ Step 1. selecting integer q from Mother or Father String

  ☞ Step 2. Performing crossover on the Mother or Father string.

  ☞ Step 3. Creating a child (daughter, son) string.



e.g.

M=(1,3,2,5,4,6)
F=(2,4,6,1,3,5)
⇨
M=(**1,3,2**|5,4,6)
F=(2,**4**,**6**,1,3,**5**)
⇨
D=(**1,3,2,4,6,5**)

M=(1,3,2,5,4,6)
F=(2,4,6,1,3,5)
⇨
M=(**1**,**3**,2,**5**,4,6)
F=(**2,4,6**|1,3,5)
⇨
S=(**2,4,6,1,3,5**)

(Existing Population)
(Pop size : 2)

(Operating CX q=3)

(Creating new string)
(Pop size : 4)

- **Mutation :** Swap two adjacent activities (considering RCPSP characteristic)

e.g. If $u \sim U(0,1)$ < Mutation probability : D=(1,**3**,**2**,4,6,5) -> D=(1,**2**,**3**,4,6,5)

*However,
it is only executed in the case of precedence feasibility, and in the case of not having precedence feasibility, the original string is maintained.

3. GA design

• GA design for RCPSP

✓ Operators

- **Selection :** Ranking method

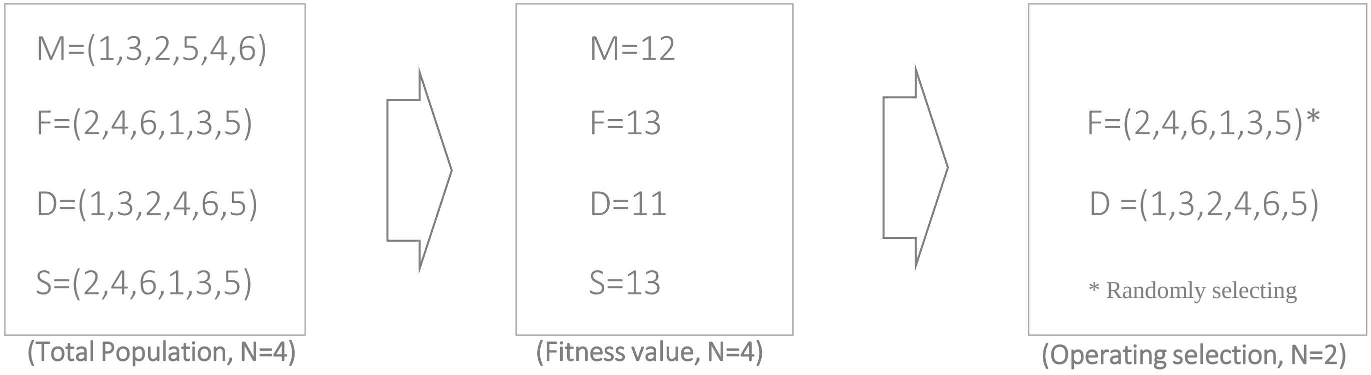
☞ Step 1. Calculating fitness values for the total population.

* Total population(2n) = Existing population(n)(Mother and Father) + New population(n)(Mother, Father, Daughter, Son)

☞ Step 2. Keeping best sting and remove the remaining ones from the population

* In case of ties, the decision is made arbitrarily.

* After applying the selection method, the population reduces from 2n to n.



M=(1,3,2,5,4,6)
F=(2,4,6,1,3,5)
D=(1,3,2,4,6,5)
S=(2,4,6,1,3,5)

(Total Population, N=4)

M=12
F=13
D=11
S=13

(Fitness value, N=4)

F=(2,4,6,1,3,5)*
D =(1,3,2,4,6,5)

* Randomly selecting

(Operating selection, N=2)

✓ Parallel GAs

- **Multiprocessing** : Calculating multiple fitness values using Python multiprocessing

* Python multiprocessing is a module in Python that supports parallel programming.
It enables the creation and management of multiple processes, allowing tasks to be executed concurrently.
This module is particularly useful for CPU-bound tasks where parallel processing can significantly improve performance.

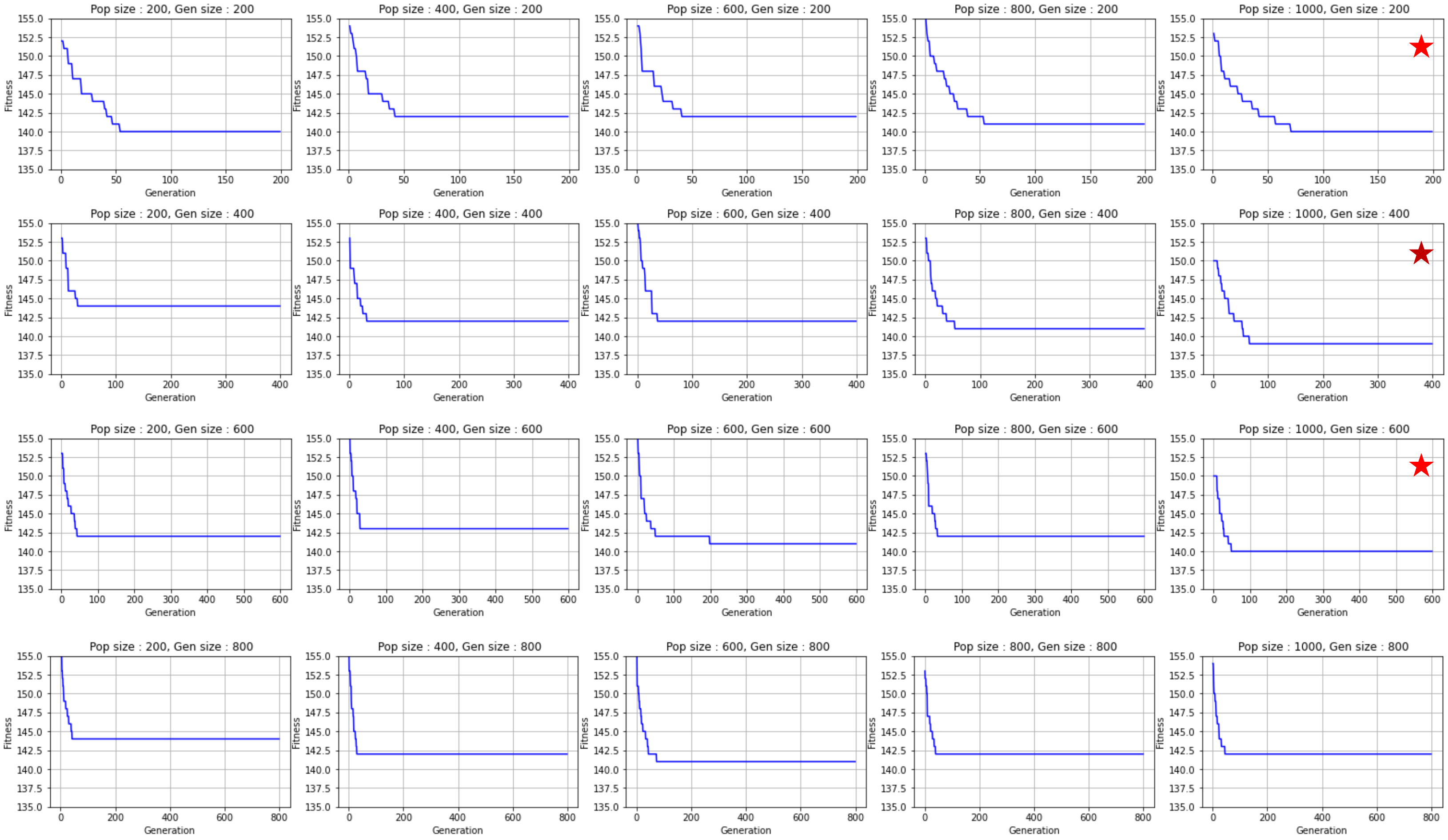
4. Performance results

4. Performance results

- GA performance results

※ The stat mark indicates a fitness value of 140 or less

✓ Case1 : Generation fixed, Population increment, Mutation probability : 5%

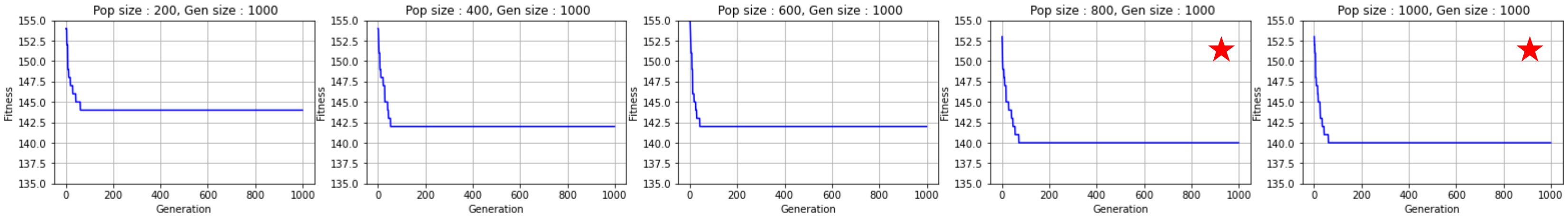


4. Performance results

- GA performance results

※ The stat mark indicates a fitness value of 140 or less

✓ Case1 : Generation fixed, <u>Population increment</u>, <u>Mutation probability : 5%</u>



In summary,

- ➢ 200 generations are sufficient to achieve satisfactory results
- ➢ There is a tendency for the fitness value to increase as the generation increases
- ➢ However, it is confirmed that the fitness value does not decrease below **140** even with an increase in generation.

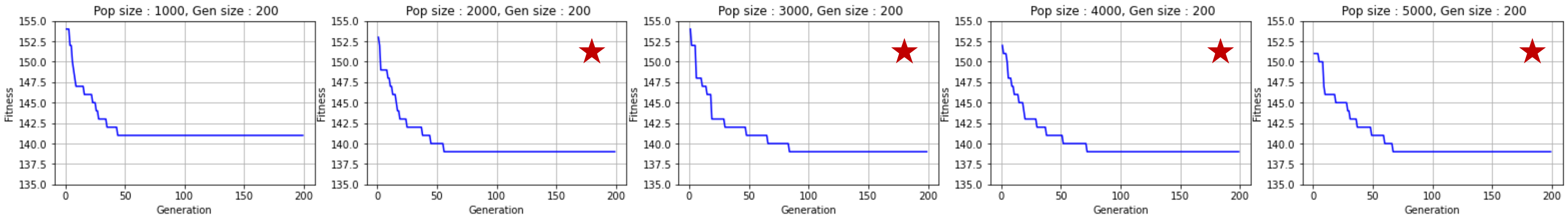
❖ To enhance the fitness value,
the effects of Population size and mutation need to be considered.

4. Performance results

- GA performance results

※ The stat mark indicates a fitness value of 140 or less

✓ Case2 : Generation fixed, Population increment, Mutation probability : 5%



In summary,

➢ As anticipated, there is an increase in fitness value with the increase in generation. However, it has been confirmed that the fitness value does not improve further beyond a population size of 3000.

➢ Still confirmed that the fitness value does not decrease below **139** even with further increases in generation.

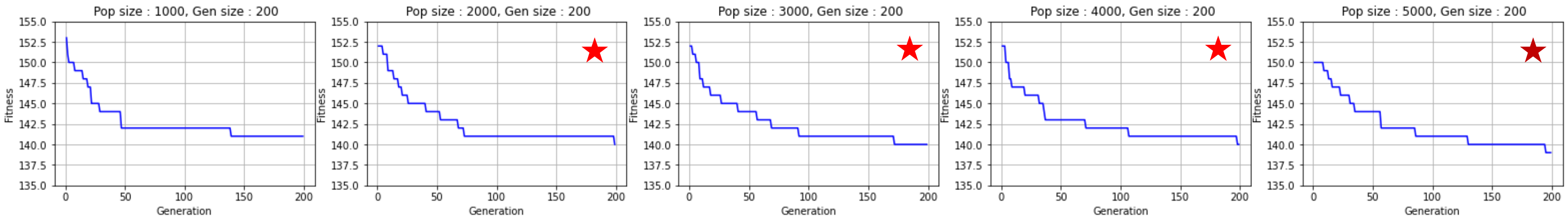
❖ To enhance the fitness value,
the effects of mutations need to be considered.

4. Performance results

- GA performance results

※ The stat mark indicates a fitness value of 140 or less

✓ Case3 : Generation fixed, Population increment, Mutation probability : 50%



In summary,

➢ It has been observed that a mutation rate of 50%(case3) results in a performance degradation compared to a mutation rate of 10%(case2)

➢ The reason for this is attributed to the fact that the effect of hindering convergence to the minimum fitness value by the fitness function is more substantial than the effect of mutation(**escaping from local optima**).

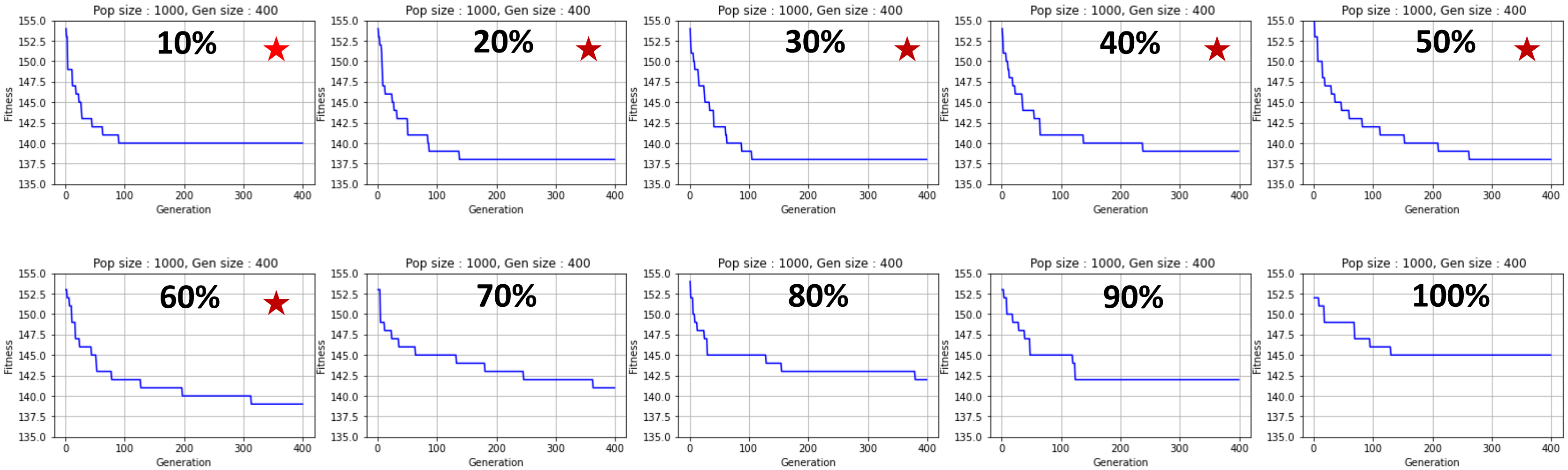
❖ Therefore, further experiments are required to establish an effective mutation probability setting.

4. Performance results

- GA performance results

※ The stat mark indicates a fitness value of 140 or less

✓ Case4 : Generation fixed, Population fixed, Mutation probability increment



➢ It has been observed that as the mutation probability increases (from 10% to 60%), the fitness value of 140 or less. (**best fitness value : 138 under Pm 30%**)

➢ It has been confirmed that the performance of the Genetic Algorithm (GA) deteriorates when the mutation probability exceeds 70% (due to excessive mutation effects, similar to case 3).

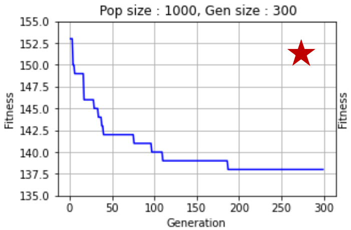
❖ The effect of quickly converging to the best fitness value has been observed at a mutation probability of 30%.

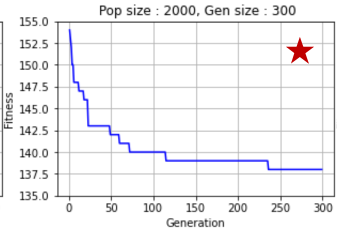
4. Performance results

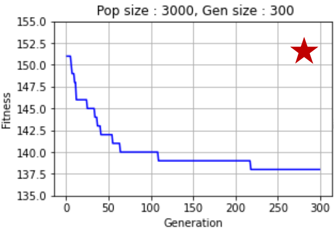
- GA performance results

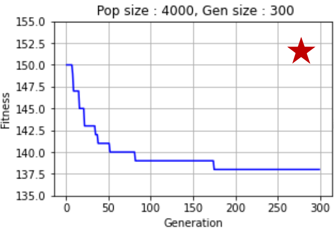
※ The stat mark indicates a fitness value of 140 or less

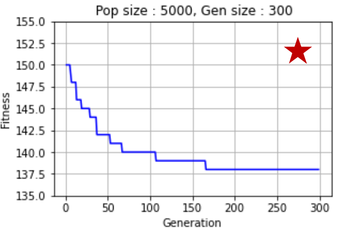
✓ Case5 : Generation fixed, Population increment, Mutation probability : 30%









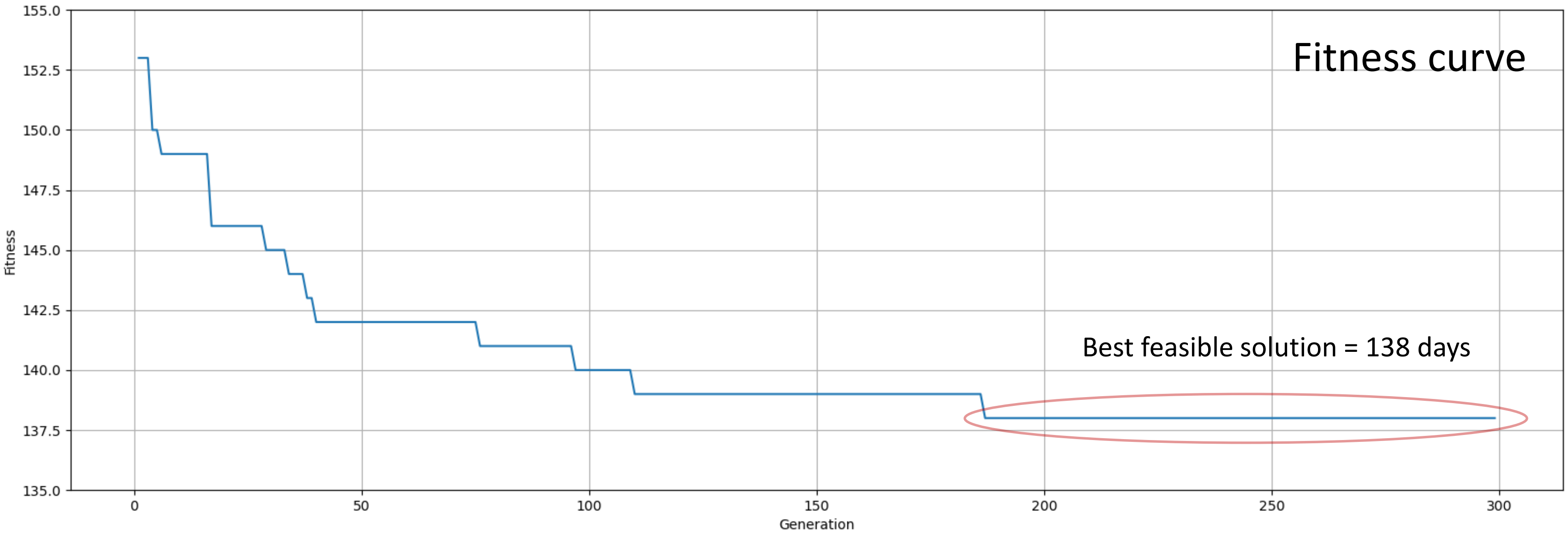


➢ Observations while increasing the population size(from 1000 to 5000) indicate that the fitness value converges to 138, and it does not decrease further.

❖ Considering the computational power, it is deemed reasonable to set the population size(1000), generation size(300), and mutation probability(30%) based on the comprehensive analysis of the previous results.

4. Performance results

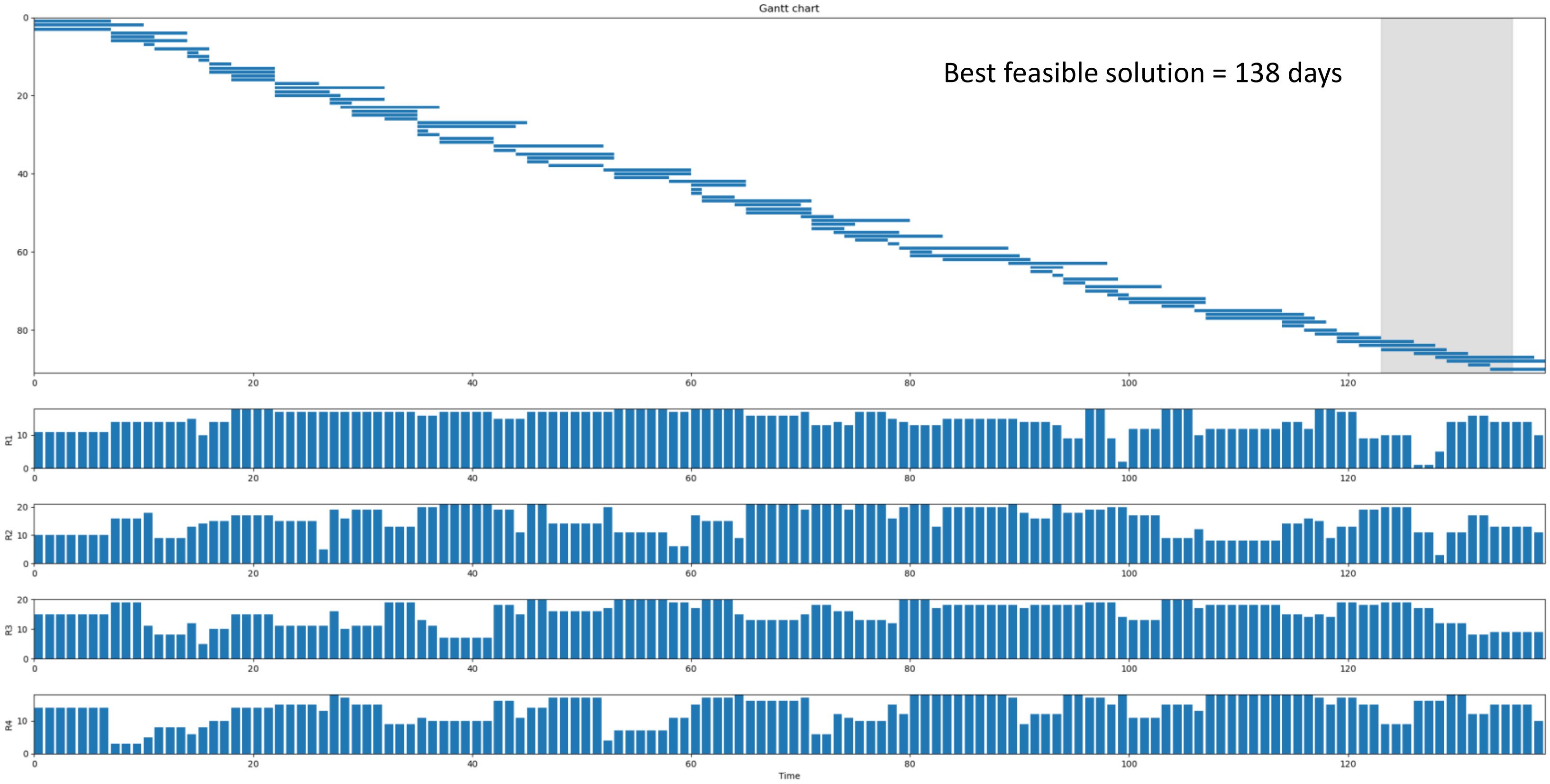
- Performance comparison results (GA vs MILP)
  - ✓ GA results using Python(Pop size : 1000, Gen size : 300, Pm : 30%)



- ➢ Fitness value = [inf, 153, 153, 153, 150, 150, 149, 149, 149, 149, 149, 149, 149, 149, 149, 149, 149, 146, 146, 146, 146, 146, 146, 146, 146, 146, 146, 146, 146, 145, 145, 145, 145, 145, 144, 144, 144, 144, 143, 143, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 141, 141, 141, 141, 141, 141, 141, 141, 141, 141, 141, 141, 141, 141, 141, 141, 141, 141, 141, 141, 141, 140, 140, 140, 140, 140, 140, 140, 140, 140, 140, 140, 140, 140, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138]

4. Performance results

- Performance comparison results (GA vs MILP)
  - ✓ GA results using Python(Pop size : 1000, Gen size : 300, Pm : 30%)



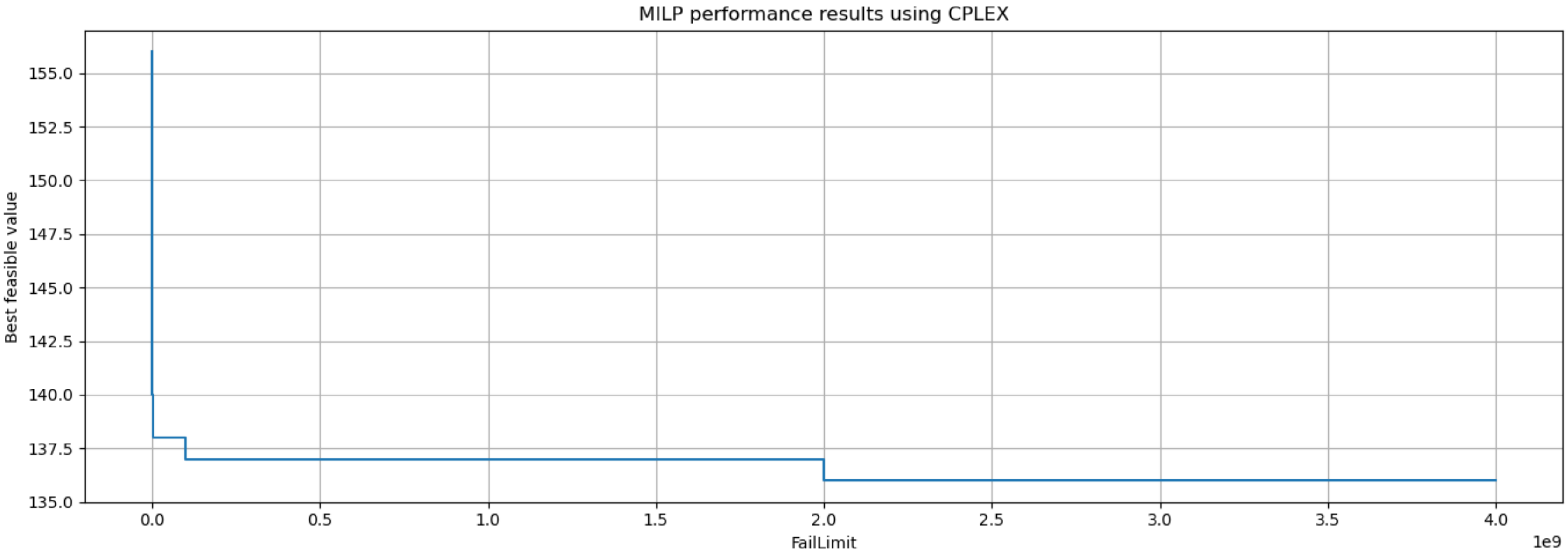
Best feasible solution = 138 days

Best feasible solution =
[0, 1, 13, 2, 15, 3, 4, 9, 5, 23, 50, 7, 8, 17, 38, 11, 14, 20, 21, 16, 24, 28, 33, 29, 12, 10, 26, 32, 27, 6, 34, 19, 41, 25, 18, 31, 22, 30, 59, 45, 44, 43, 53, 61, 39, 71, 37, 49, 36, 35, 40, 76, 55, 47, 42, 60, 46, 58, 51, 63, 65, 48, 57, 66, 52, 68, 64, 73, 67, 77, 72, 62, 80, 70, 81, 54, 78, 79, 69, 75, 56, 85, 82, 84, 86, 87, 74, 88, 83, 90, 89, 91]

4. Performance results

- Performance comparison results (GA vs MILP)

✓ MILP results using Python - CPLEX



※ **FailLimit** parameter specifies the maximum number of failures that the solver tolerates before it stops searching for a solution.

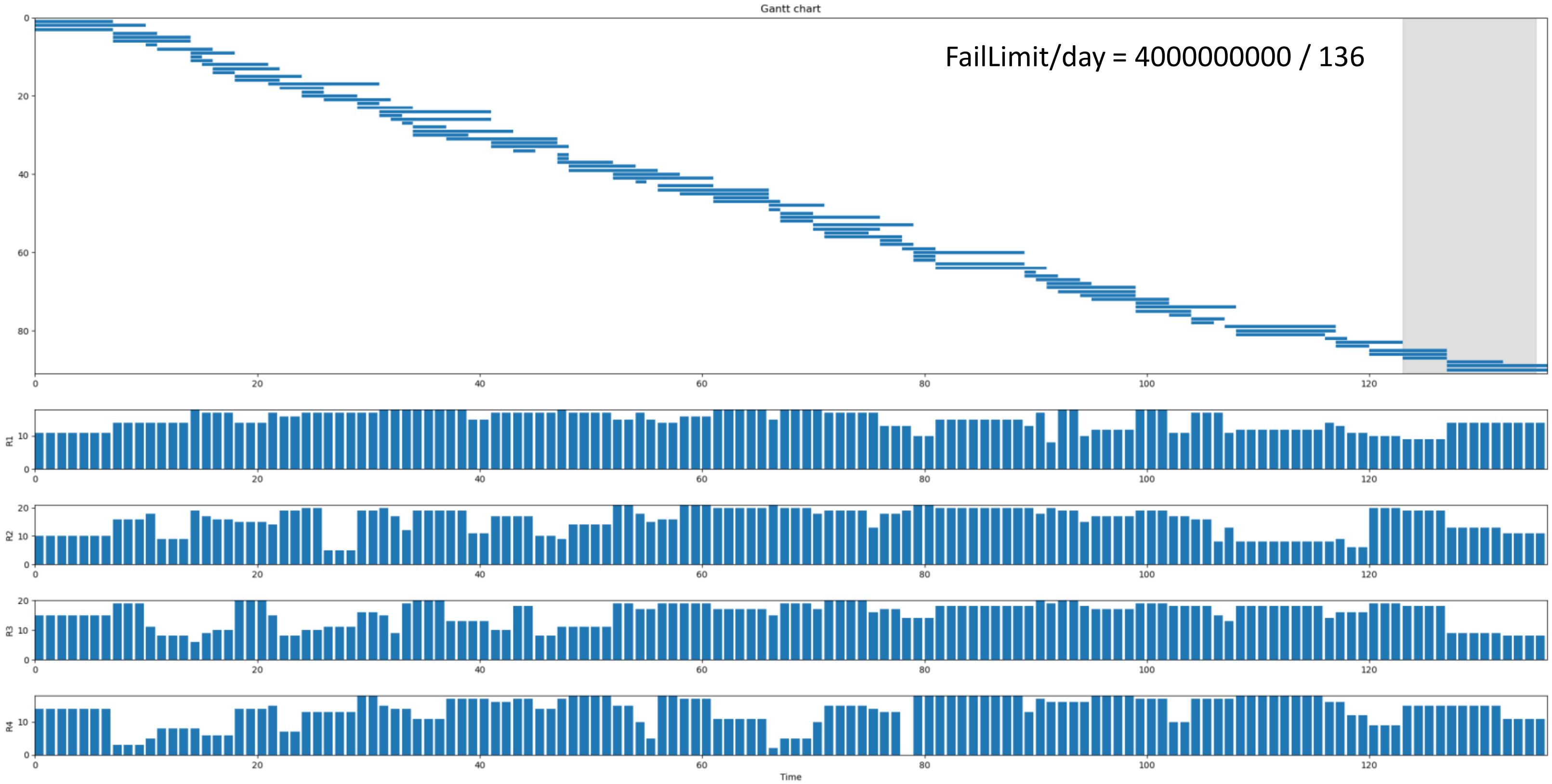
➢ FailLimit = [100, 500, 1000, 5000, 10000, 50000, 100000, 200000, 500000, 1000000, 10000000, 100000000, 1000000000, 2000000000, 3000000000, 4000000000]

➢ Best feasible value = [156, 152, 152, 151, 150, 148, 146, 143, 141, 140, 138, 138, 137, 137, 136, 136]

4. Performance results

- Performance comparison results (GA vs MILP)

  ✓ MILP result using Python - CPLEX



FailLimit/day = 4000000000 / 136

Best feasible solution =
[0 1 2 3 13 9 4 8 15 11 16 5 19 20 23 7 50 21 24 43 17 14 33 12 10 26 22 38 28 30 59 27 29 18 41 45 31 32 37 34 36 71 48 53 39 6 35 49 25 44 55 60 40 63 47 76 42 73 61 46 52 66 77 65 58 68 57 56 69 70 80 74 67 62 51 72 83 54 64 81 78 79 84 86 75 85 87 82 90 88 89 91]

5. Conclusion & future plan

5. Conclusion & future plan

- Overall analysis results

  ✓ With the increase in generation, population and mutation probability, there is an observed tendency for the fitness value to increase. However, it falls short of reaching the performance of MILP

  ※ The currently designed GA is judged to have difficulty deviating from the local optimal area

  ✓ However, the fact that GA shows a difference of only 2 days compared to CPLEX in solving the challenging RCPSP problem with 90 activities indicates the effective performance of GA.

  ※ when applying ALNS, a value of 141 is found (ref . Muller, LF. 2009)

- Future improvement and enhancement plans

  ✓ Improvement of GA method (considering variety of operators and characteristics of RCPSP)

  ✓ Comparison of performance with other metaheuristic models (such as Adaptive Large Neighbourhood Search(ALNS) etc)

  ※ ALNS is characterized by its ability to dynamically adjust the neighborhood structure during the search process, aiming to efficiently explore the solution space.

Thanks