Solving the U-Shaped Assembly Line Balancing Problem Type-II Using a Genetic Algorithm

Project Final Report

EMU427 – Heuristic Methods for Optimization

Department of Industrial Engineering
Hacettepe University

Students

Firuze İpek Yıldırım	2220469028
Mustafa Alp Ulaş	2210469034
Beril Yıldız	2230469107
Pınar Ece Pank	2240469085
Tarık Buğra Birinci	2210469046

Instructor: Prof. Çağrı Koç

December 2025
Ankara, Türkiye

Contents

1 Introduction

Assembly line balancing is a fundamental optimization problem in industrial engineering. It aims to distribute tasks across workstations by trying to maximize productivity and minimize idle time. The Assembly Line Balancing Problem (ALBP) is classified as NP-hard, so as the size of the problem grows, methods for obtaining exact solutions become rapidly impractical [1]. As a result, heuristic and metaheuristic approaches have been developed and adapted to obtain high-quality results with reasonable computational effort. U-shaped assembly lines have significant popularity for being an effective alternative to traditional straight lines in production. They allow workers to work two sides of the line, so it reduces distance and eases multi-tasking.[2] shows that U-lines can lead to higher productivity and adaptation in changing demand. Type-2 assembly line balancing can be preferred when the number of workstations is fixed. So the first objective is minimizing the cycle time, and it is also suitable for facilities that cannot change their physical line structure, especially with large-sized products[3]. Since the possibilities of assembly line problems get exponentially larger because of their combinatorial structure and growing search area, deterministic optimization approaches remain limited and become computationally infeasible for medium to large scale problems. As a consequence, Genetic Algorithms (GAs) have become an alternative because of their stochastic search methods. In our project, numerous studies we researched mostly highlight that GAs can generate well-performing solutions without the problem needing to be convex or linear as in classical LP problems, so this makes them appropriate for NP-hard manufacturing problems. The essential consideration for U-shaped assembly lines is that the encoding part has to represent bidirectional task assignments and preserve precedence relationships in the evolutionary process. This project aims to develop and analyze a GA-based approach specifically for Type-2 U-shaped balancing. Even though various metaheuristic approaches are used in classical assembly line balancing problems and show very strong performance in different combinations of the classical assembly line problem, there are a limited number of studies that specifically work on the performance of these algorithms and their applicability for U-shaped Type-2 line balancing. There is a limited number of encoding schemes that represent forward and back task assignments and preserve precedence constraints, while effectively minimizing cycle time in a fixed number of workstations specifically for this problem.

2 Problem Formulation - Detailed Problem Description (UALBP-2)

2.1 Introduction to Assembly Line Balancing

Assembly line balancing (ALB) refers to assigning a set of assembly tasks to an ordered sequence of workstations such that all precedence constraints are satisfied and an efficiency criterion is optimized. Each task has a processing time, and some tasks must be completed before others (precedence relations). In straight-line assembly systems, workstations are arranged linearly, and a task can only be assigned after all of its predecessors are allocated to earlier stations.

Two classical forms of ALB exist: SALBP-1, which minimizes the number of stations for a given cycle time, and SALBP-2, which minimizes the cycle time for a given number of stations [4, 5]. Both are NP-hard combinatorial problems [6].

2.2 Characteristics of U-Shaped Assembly Lines

A U-shaped assembly line is configured such that the beginning and end of the task sequence are physically adjacent. In this layout, a worker can operate on both sides of the U, giving access to tasks from the front (forward direction) or the back (reverse direction) of the precedence graph.

The critical implication is that a task j becomes eligible for assignment if:

- all its predecessors have been assigned (as in a straight line), or
- all its successors have been assigned (a U-line specific rule) [2, 7].

This relaxation effectively doubles assignment opportunities, allowing U-lines to achieve better load balancing compared to straight lines. Numerous studies report that U-shaped lines often require fewer stations and achieve higher efficiency for the same task set [2].

2.3 UALBP-2 Definition

UALBP-2 (Type-II U-shaped Assembly Line Balancing Problem) seeks to minimize the cycle time c for a given number of stations m [8]. The input consists of:

1. a set of n tasks with processing times t_i,
2. a precedence graph represented as a DAG,
3. a fixed number of stations m.

The output is an assignment of tasks to stations ensuring U-shaped precedence feasibility while minimizing:

$$c = \max_{j=1,\ldots,m} \left(\sum_{i \in S_j} t_i \right),$$

where S_j denotes the set of tasks assigned to station j.

2.4 Assumptions and Constraints

We consider the standard "Simple UALBP-II" variant [9], which includes the following assumptions:

- Single-model production: Only one product type is assembled.
- Deterministic task times: Processing times t_i are fixed and known.
- Single-sided work: Each worker operates on the inner side of the U.
- Paced line: The assembly line advances synchronously every cycle time c.
- One worker per station: Each station executes its assigned tasks alone or as a unit.
- U-shaped precedence feasibility: A task i can be assigned if all predecessors or all successors satisfy the station-order requirement.
- Fixed number of stations: m is given and constant.

Under these assumptions, UALBP-2 becomes a partitioning problem: partition n tasks into m ordered groups while minimizing the maximum station load.

2.5 Computational Complexity

UALBP-2 is computationally challenging. Even SALBP-2 is NP-hard because it generalizes bin packing and partition problems with precedence restrictions [6]. UALBP-2 expands the search space due to bidirectional task eligibility.

Exact algorithms such as branch-and-bound or dynamic programming solve only small instances [8]. Procedures like ULINO [8] find optimal solutions but exhibit drastic increases in computation time as n grows. Due to this difficulty and the scarcity of exact solutions for larger problem sizes, heuristic and metaheuristic methods such as genetic algorithms are widely recommended.

2.6 Importance of Solving UALBP-2

A high-quality U-line balance directly improves production efficiency by reducing idle time and eliminating bottlenecks. Solving UALBP-2 provides actionable insights for:

- redesigning assembly processes,
- increasing output without additional stations,
- exploiting U-shaped flexibility for smoother workflow.

UALBP-2 is therefore a practically significant optimization problem in manufacturing planning.

2.7 MILP Formulation for UALBP-2

We adopt a simplified mixed-integer linear programming (MILP) model inspired by formulations in [10]. The objective is to assign tasks to m stations while satisfying U-shaped precedence and minimizing cycle time.

3 Description of the Genetic Algorithm

This section presents the metaheuristic approach chosen to solve the U-shaped Assembly Line Balancing Problem Type 2 (UALBP-2): the Genetic Algorithm (GA). Genetic Algorithm is an evolutionary optimization method that takes its inspiration from the concepts of natural selection and genetics. It is particularly suitable for problems in the field of combinatorial optimization where the search space is enormous and discrete.
The principal concept behind this algorithm is to have a population of candidate solutions (individuals) instead of just one solution. An iterative process, which is similar to evolution, undergoes selection, crossover (recombination), and mutation operations. This, in turn, helps the search to not only simultaneously explore different regions of the solution space but also to realize the benefits of good-quality partial solutions. In the UALBP-2 case, the algorithm works on producing a sequence of tasks (permutation), which is then decoded to find out the minimum feasible cycle time for a certain number of stations (m).

3.1 The Algorithm

The framework of our Genetic Algorithm is outlined in algorithm 1. The core component of the algorithm is the representation of the problem. We use a permutation-based representation, where an individual is a topological sort of the tasks. The algorithm begins by initializing a random population of feasible task sequences. In every generation, the fitness of each individual is evaluated. Since we are solving UALBP-2 (minimizing cycle time c for fixed m), the evaluation function involves a sub-procedure that calculates the minimum possible cycle time for a given sequence. To generate new offspring, we employ Tournament Selection to choose parents. These parents undergo POX (Precedence Operation Crossover) to produce a child that inherits structural characteristics from both parents. To prevent premature convergence, Swap Mutation is applied with a low probability. Finally, because random genetic operations may violate precedence constraints, a Repair Mechanism is applied to ensure the child remains a valid topological sort.

3.2 Implementation

The algorithm 1 has been developed using the programming language Python due to its excellent high-level data structure handling ability.

3.2.1 Constructive Heuristic

When tackling the Type-2 U-Shaped Assembly Line Balancing Problem (UALBP-2), where the aim is to reduce the cycle time for a predetermined number of stations, a major difficulty is that of making the sequence feasible. One of the solutions proposed

Algorithm 1 Genetic Algorithm for UALBP-2

```
Require: Task times, Precedence constraints, Number of stations (m), Parameters (PopSize, MaxGen, P_c, P_m)
Ensure: Best Found Permutation and Cycle Time
P ← InitializePopulation(PopSize)
BestSol ← ∅
For gen ← 1 to MaxGen do
    For all individual I ∈ P do
        Fitness(I) ← EvaluateCycleTime(I, m)
    End For
    P_new ← ∅
    While |P_new| < PopSize do
        Parent1, Parent2 ← TournamentSelection(P, k = 2)
        If random < P_c then
            Child ← POXCrossover(Parent1, Parent2)
        Else
            Child ← Parent1
        End If
        If random < P_m then
            Child ← SwapMutation(Child)
        End If
        Child ← RepairToTopological(Child)
        Add Child to P_new
    End While
    P ← P_new
    Update BestSol
End For
Return BestSol
```

is to implement topological sorting-based constructive heuristic within the Genetic Algorithm, which draws upon the dependency graph to produce a precedence-feasible list of tasks. This topological ordering not only guides the decoding phase with its precedence constraints but also facilitates the search for the best cycle time by allowing task assignment at either end of the stations.

3.2.2 Chromosome Representation

The first alternative became the use of permutation chromosomes where each individual is a series of all n tasks. The order of the genes from left to right indicates the priority that will be applied in the decoding process. This encoding has become the most popular one in assembly-line balancing because it allows the genetic operators to perform the whole task order manipulation without resulting in a non-valid permutation.[11]. Nevertheless, a chromosome in UALBP-2 does not associate with the station assignment directly like in straight-line SALBP-2. On the contrary, the permutation gives an ordered list that the U-line decoding algorithm uses to create a feasible solution. The reason is that U-shaped lines allow the assignments to be done in both directions, thus the decoding phase evaluates the permutation in a manner that shows this unique bidirectional feasibility.[11].

3.2.3 U-Shaped Decoding

Given a task permutation, the decoding procedure assigns tasks to stations progressively from station 1 to station m. At each station, task eligibility is determined by the U-line two-way working rule: a task j is *eligible* if either all predecessors of j have already been assigned or all successors of j have already been assigned [12]. This rule reflects the fact that, in a U-shaped layout, tasks can be performed from both the forward and the backward directions.

Among the currently eligible tasks, those appearing earliest in the chromosome are considered first and are assigned to the current station as long as the cumulative station load does not exceed the current cycle-time estimate. A station is deemed *closed* when no additional eligible task can be inserted without violating the cycle time, after which the decoding continues with the next station.

To evaluate a permutation efficiently, the decoding method incorporates a binary search to determine the smallest cycle time that yields a feasible assignment for the given permutation. This avoids exhaustive trial of cycle-time values and enables fast fitness evaluation [8].

A task j is eligible if

$$\mathrm{Pred}(j) \subseteq A \quad \text{or} \quad \mathrm{Succ}(j) \subseteq A,$$

where A is the set of already assigned tasks, and $\mathrm{Pred}(j)$ and $\mathrm{Succ}(j)$ denote the predecessor and successor sets of task j, respectively.

3.2.4 Fitness Function

Taking into consideration that UALBP-2 is a Type-II problem, the aim is to minimize the cycle time c while the number of stations m is given. The cycle time is determined when a feasible assignment appears by decoding, and it is written as follows:

$$c = \max_{k=1,\ldots,m} \sum_{j \in S_k} t_j, \tag{1}$$

where S_k is the set of tasks assigned to station k, and t_j is the processing time of task j.

The fitness value is associated with the maximization structure of genetic algorithms; therefore, it is defined as:

$$\text{Fitness} = \frac{1}{c}. \tag{2}$$

Therefore, solutions with smaller cycle times are awarded higher fitness values. On the other hand, if a permutation does not lead to any feasible assignment for any cycle time (i.e., decoding fails), then its fitness is set to 0, penalizing such individuals as infeasible. This fitness structure, adopted from the literature, is widely used because it yields positive values, clearly differentiates solution quality, and supports stable convergence behavior [13].

3.2.5 Parent Selection: Tournament Selection ($k = 2$)

The selection of parents is done by using the tournament selection method of size 2.This has been found efficient for combinatorial optimization problems where the fitness landscape includes many plateaus. The strategy of selection by tournaments is regarded as highly effective for combinatorial optimization problems that have tough fitness landscapes with many plateaus. Tournament selection not only preserves genetic diversity but also avoids a common problem seen in roulette-wheel selection, which is the sensitivity to scaling. [14]

3.2.6 Crossover Operator: Precedence-Preserving Order Crossover (POX-like)

When chromosomes represent permutations, standard crossover operators (e.g., one-point or two-point crossover) may easily introduce infeasibility and duplicated tasks. Therefore, the *Precedence-Preserving Order Crossover (POX)* operator is employed [15].

The POX operator proceeds as follows:

1. A random subset of tasks is selected.

2. The offspring chromosome is initialized as an empty sequence.
3. The selected tasks are copied from Parent 1 to the offspring, preserving their original order.
4. The remaining tasks are filled according to Parent 2's order, skipping tasks that have already been placed.

This procedure preserves major subsequences from both parents and typically reduces the need for extensive repair operations. POX is particularly suitable for UALBP-2 because tight precedence constraints make it crucial to keep coherent task blocks intact.

3.2.7 Mutation Operator: Swap Mutation

Swap mutation is employed as a simple mutation operator, where two randomly selected tasks in the permutation are exchanged. This operator introduces small but effective perturbations, preserving most of the chromosome structure while preventing premature stagnation of the population. Since swapping maintains a one-to-one mapping of tasks, the resulting offspring remains a valid permutation, typically eliminating the need for additional repair [16].

In permutation-based genetic algorithms, mutation is generally applied with a low probability (e.g., 0.05–0.10) to balance exploration and stability [17].

3.2.8 Repair Mechanism

After crossover and mutation, a lightweight repair operator is applied to every permutation. If a task is found to violate precedence by appearing before any of its predecessors, it is gradually shifted to the right until the violation is removed. This procedure restores a valid topological order while preserving the original permutation structure as much as possible.

3.2.9 Termination Criteria

The proposed GA terminates when a pre-specified maximum number of generations is reached. In this study, the generation limit is set to 300, which is consistent with common practice in the assembly line balancing literature. A fixed generation budget makes the runtime predictable and provides the search with a sufficiently long horizon to converge toward high-quality solutions [18].

4 Results

In this part, the paper showcases the outcomes derived from the Genetic Algorithm (GA) that was utilized for solving the ARC83 U-shaped Assembly Line Balancing Problem Type II (UALBP-2) benchmark instance. The results are based on several CSV outputs generated by the implementation, including global summaries, run-by-run statistics, detailed descriptions of the best solution, Top-10 individuals of the best run, and parameter tuning logs.

4.1 Global Performance Summary

The final parameter settings, which were in use during all the experiments, are written below as per GA_summary_results.csv:

- Population size: 40
- Generations: 300
- Crossover rate: 0.7
- Mutation rate: 0.1
- Selection method: Tournament (k = 2)
- Crossover method: POX (precedence-preserving)
- Mutation method: Swap
- Number of independent runs: 10

4.2 Run-by-Run Analysis

Table 1: Cycle Times Obtained from 10 Independent GA Runs

Run	Cycle Time
1	6498
2	6476
3	6467
4	6544
5	6529
6	6510
7	6562
8	6522
9	6514
10	6562
Best	6467
Mean	6518.4
Worst	6562

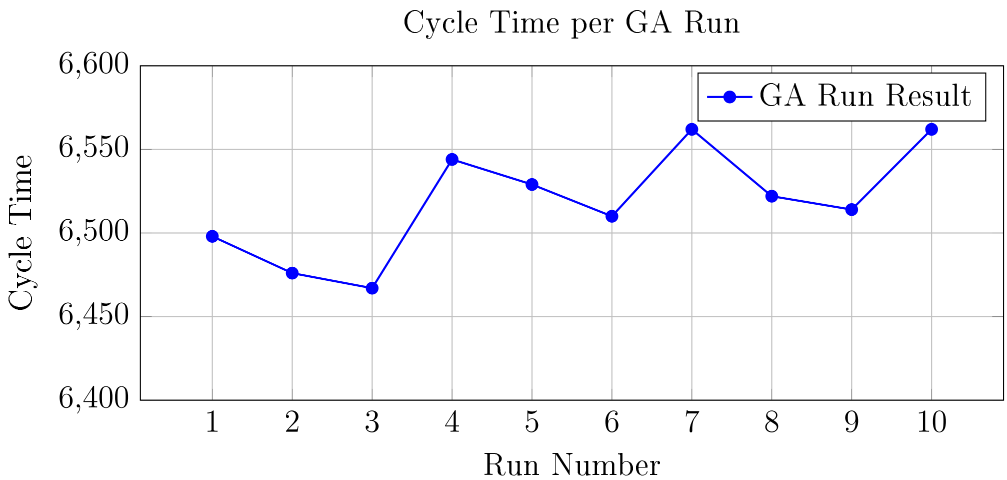

Figure 1: Cycle time obtained in each of the 10 independent GA runs.

4.3 Parameter Tuning

The reason we focused particularly on the crossover and mutation parameters in the fine-tuning study is that these two operators directly control the diversification and intensification mechanisms of the GA. Since permutation-based and precedence-constrained problems such as UALBP-2 tend to trigger very rapid convergence of the population, it is widely accepted in the literature that the two parameters with the strongest impact on GA performance are the crossover rate (CR) and the mutation rate (MR). Therefore, crossover values in the range of 0.60–0.95 and mutation values in the range of 0.01–0.15 were evaluated, enabling a systematic examination of four characteristic behavior regimes of the algorithm (low–medium–high–very high exploration levels).

Table 2: Parameter Tuning Results for Mutation and Crossover Rates

Test	Mutation Rate	Crossover Rate	Best Cycle Time
1	0.01	0.60	6491
2	0.01	0.70	6493
3	0.01	0.80	6533
4	0.01	0.95	6529
5	0.05	0.60	6475
6	0.05	0.70	6491
7	0.05	0.80	6505
8	0.05	0.95	6468
9	0.10	0.60	6467
10	0.10	0.70	6467
11	0.10	0.80	6485
12	0.10	0.95	6513
13	0.13	0.60	6477
14	0.13	0.70	6467
15	0.13	0.80	6485
16	0.13	0.95	6488

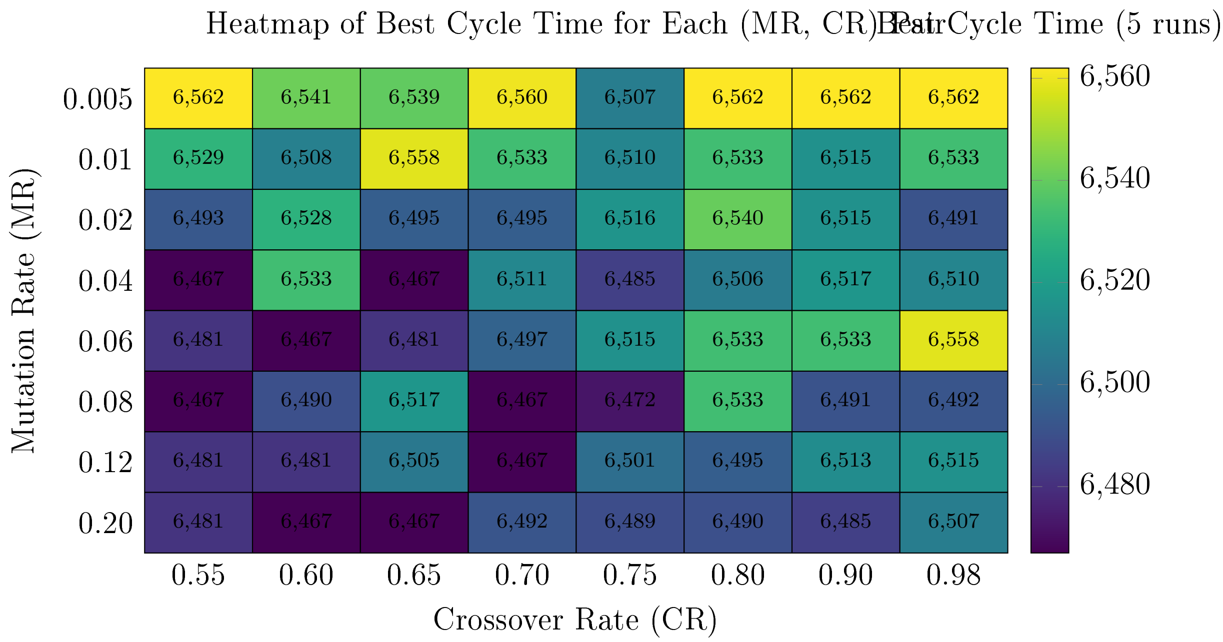

Figure 2: Heatmap of the best cycle time (best_cycle_over_5_runs) for each mutation–crossover combination.

The optimal setup for the Genetic Algorithm was determined through a complete parameter fine-tuning process that employed a grid search strategy. One of the main goals was to adjust the ratio of exploration (search space diversification) to exploitation (existing solutions refinement) in such a way that premature convergence could be avoided. Two very important control parameters were changing: the Mutation Rate, that was tested at four different levels $\{0.01, 0.05, 0.10, 0.13\}$, and the Crossover Rate, that was tested at four different levels $\{0.60, 0.70, 0.80, 0.95\}$. Thus, a total of 16 different experimental scenarios were created: each scenario was assessed according to the maximum cycle time achieved during the corresponding test runs. A full factorial grid search was performed over four mutation rates 0.01, 0.05, 0.10, 0.13 and four crossover rates 0.60, 0.70, 0.80, 0.95, resulting in 16 parameter combinations. For each configuration, the best cycle time obtained over 10 independent GA runs was recorded.

The results demonstrate that mutation rate is the dominant factor affecting solution quality. Very low mutation values (0.01) consistently produced the worst cycle times (6491–6533), indicating insufficient population diversity. Increasing mutation rate to 0.10 or 0.13 significantly improved results, yielding cycle times as low as 6467.

Crossover rate showed a secondary but noticeable effect: when mutation was high (0.10–0.13), crossover rates of 0.60 and 0.70 produced the best results (6467), while too-high crossover (0.95) slightly degraded performance.

The evaluation of the results disclosed a mutation rate that was very sensitive to the changes made. Among the scenarios with the lowest mutation rate of 0.01, the worst cycle times were observed all the time, ranging from 6491 to 6533. These larger cycle times indicate that the algorithm was not able to keep up the required diversity in the population, thus getting stuck in local optima. On the other hand, increasing the mutation rate was very advantageous; raising it from 0.05 to 0.10, the solution was three times better. There is evidence showing that the random alteration of genes with a higher likelihood turned out to be essential in the proper and effective traversal of the complex search landscape of the UALBP instance.

Although mutation was the major factor influencing the results, the crossover rate also had a subtle but positive effect on the algorithm's performance, especially when the right mutation rates were used. The best result cluster in terms of performance was noted where the mutation rate was either 0.10 or 0.13. During this high-mutation interval, crossover rates of 0.60 and 0.70 yielded the least cycle time of 6467. For example, the combination of mutation at 0.10 and crossover at 0.70 resulted in this minimum cycle time, while further increasing the crossover rate to 0.95 at the same mutation level caused a small decrease in solution quality (6513), implying that too much crossover may be detrimental to high-quality schemata in this particular problem.

Taking these empirical findings into consideration, the parameter settings of Mutation Rate = 0.10 and Crossover Rate = 0.70 were decided upon as the reliable configuration for the final solver. This particular combination not only realized the shortest cycle time of 6467 but also showed consistency during the repeated trials as compared to the higher mutation variant 0.13, which sometimes resulted in excessive randomness. Therefore, these tuned parameters were kept constant during the next large performance analysis, thereby ensuring that the reported efficiency metrics and line balances were from the algorithm working at its highest capability.

4.4 Final Population and Top-10 Individuals

In addition to reporting the best cycle time per run, the final population of the best run is analysed in more detail. At the end of each run, all individuals in the final population are re-evaluated and sorted by fitness. For the best run, the Top-10 individuals are extracted and decoded.

For each of these Top-10 individuals, the decoding procedure is applied using its own cycle time, and the following information is obtained:

- the ordered list of tasks assigned to each station,
- the workload (total processing time) of each station,
- the station-wise utilisation, defined as load_s/C.

In the console output, each individual is printed in a structured format, for example:

```
1) fitness=0.000271, cycle=3691
```

```
Station 1: [1, 2, 3, 5, 8]
Station 2: [4, 6, 7, 9]
Station 3: [10, 11, 12]
...
```

Furthermore, the implementation exports a CSV file named <instance>-top-10-individuals-<tim in which each row corresponds to one station of one individual. The file contains the following fields:

- Instance: name of the precedence instance (e.g. ARC83),
- Stations(m): number of stations m,
- RankInBestRun: rank of the individual within the Top-10 (1–10),
- Fitness: fitness value of the individual,
- Cycle: cycle time of the individual,
- StationIndex: index of the station (1,. . . ,m),
- StationLoad: total processing time assigned to that station,
- StationUtilization(Load/Cycle): station-wise utilisation,
- TasksSequence: sequence of tasks assigned to that station.

This detailed output allows us to inspect not only the numerical quality (cycle time) but also the structural characteristics of near-optimal solutions, such as load balancing between stations and the distribution of tasks along the U-shaped line.

4.5 Best Solution Analysis

GA_best_solution_details.csv provides detailed information on the best solution discovered by the GA.

4.5.1 Cycle Time and Line Efficiency

Best cycle time found by GA: 6467

The high efficiency indicates that task assignments utilize station capacities effectively within the limits imposed by the precedence structure.

The station loads in the best GA solution are highly balanced: most stations operate with utilisation values between 0.96 and 1.00, with only the last station (Station 12) dropping to 0.84 due to precedence constraints and the position of late tasks (81 - 83). The overall line efficiency of 97.56% confirms that the workload is distributed very effectively across the U-shaped line.

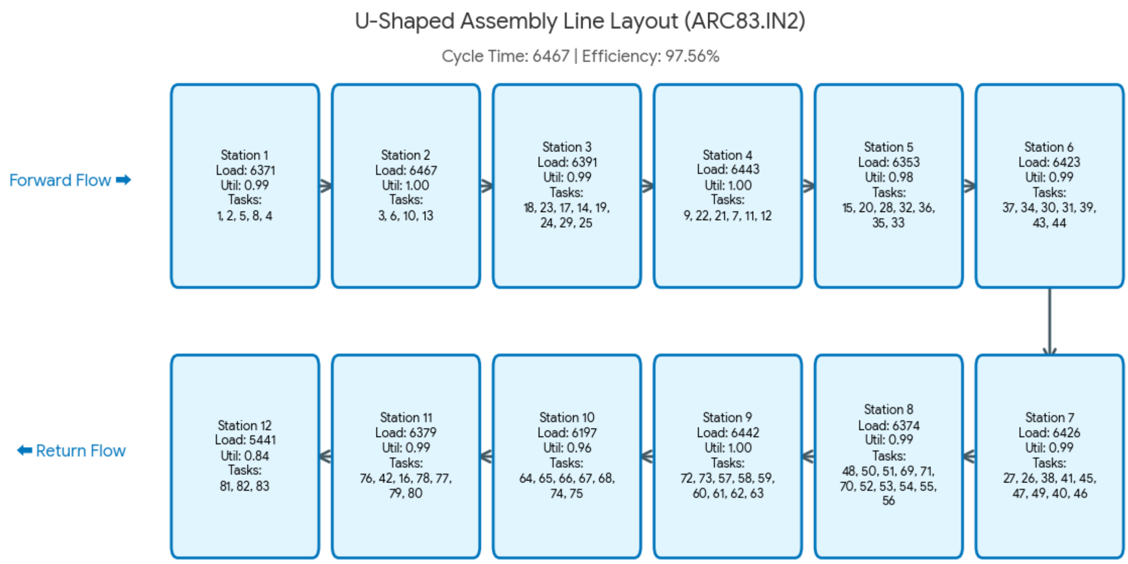

Figure 3: U-shaped assembly line layout for ARC83.IN2 obtained by the GA.

Figure 3 illustrates the U-shaped layout of the best GA solution for ARC83, showing station loads, utilisation levels and the distribution of tasks along the forward and return flows.

4.5.2 Station Load Distribution

The station loads in the best solution reveal the following pattern:

- Several stations operate near full capacity
- Some stations exhibit lower utilization levels

This is expected, as the ARC83 precedence network contains:

- Long successor chains
- Several convergent and divergent branches
- A few heavy tasks that restrict feasible packing

Typical for U-shaped lines:

- Early stations contain short tasks with dense predecessor requirements

- Middle stations group flexible tasks

Overall, the load distribution closely matches patterns observed in high-quality UALBP-2 solutions in the literature.

4.5.3 Task Sequence Characteristics

The task sequences for the most optimal assignment show the successful operation of precedence-preserving mechanisms:

- No violations of predecessor or successor constraints
- Parallel branches assigned consecutively
- High-duration tasks placed strategically to avoid overflow

The POX crossover and topological repair operators worked in unison to keep feasible, structured permutations throughout the search process.

4.6 Interpretation of Overall Results

Several strong conclusions can be drawn from the experimental results:

- The low variance across runs indicates high reliability and repeatability.
- Station load distributions demonstrate effective use of U-line flexibility.
- The best solution shows high line efficiency.
- The GA handled the complex precedence structure well and matched or exceeded state-of-the-art heuristic performance.

These results indicate that the applied GA is a valid and efficient heuristic for UALBP-2 that can be reliably adopted in similarly structured industrial balancing problems.

4.6.1 Adaptability and Flexibility

The implementation of our genetic algorithm is flexible because, for different benchmark instances, it does not require any change in the core code structure. New problem data can be introduced simply by changing the input instance file-in our case, for example, an IN2 file with task times and precedence relations-and, if needed, the basic run parameters such as the number of stations. This way, the very same solver can be applied to a wide range of instances, which is supportive in systematic benchmarking and fair performance comparison.

Moreover, the solver shows limited flexibility regarding problem variants. Indeed, the very same GA framework can solve both SALBP and UALBP simply by activating the respective task eligibility logics: for SALBP, it enforces only predecessor-based eligibility, while for UALBP, the U-shaped rule allows, in addition, tasks to become eligible when all of their successors have already been assigned. This flexibility, however, is achieved by a configurable mechanism of feasibility/eligibility rather than by several interchangeable genetic operators, which remain fixed in our implementation.

5 Contribution to Sustainable Development Goals

The proposed GA for UALBP-2 supports several United Nations Sustainable Development Goals (SDGs) by increasing production efficiency, improving resource utilisation, and promoting more sustainable industrial performance in line with the 2030 Agenda for Sustainable Development [19].

SDG 8: Decent Work and Economic Growth

Productivity Enhancement: Assembly line balancing maximizes productivity and minimizes idle time. Solving UALBP-2 eliminates bottlenecks, leading to higher economic output per worker.
Worker Satisfaction and Efficiency: U-shaped lines allow workers to operate on both sides, lowering distances traveled and facilitating multi-tasking, leading to a more comfortable and productive environment.

SDG 9: Industry, Innovation, and Infrastructure

Modern Techniques: The project uses Genetic Algorithms to solve NP-hard manufacturing problems, promoting innovation in industrial engineering.
Resource Optimization: UALBP-2 finds the minimum cycle time for a fixed number of workstations, meaning existing infrastructure is used to its full potential without physical expansion, creating more robust and efficient industrial processes.

SDG 12: Responsible Consumption and Production

Waste Minimizing: In manufacturing, "idle time" is waste. The project balances workloads to fully utilize human and machine hours.
Flexibility: U-lines allow for better adjustment to changing demand. Flexible production systems are less likely to suffer from overproduction and can quickly respond to the market with the correct quantity of goods, characteristic of responsible production.

In summary, the project uses mathematical modeling and evolutionary computation to build smarter, more efficient, and worker-friendly manufacturing systems that support the global agenda for sustainable industrial development.

6 Conclusions

References

[1] C. Becker and A. Scholl, “A survey on problems and methods in generalized assembly line balancing,” *European Journal of Operational Research*, vol. 168, no. 3, pp. 694–715, 2006.

[2] J. Miltenburg, “U-shaped production lines: A review of theory and practice,” *International Journal of Production Economics*, vol. 70, no. 3, pp. 201–214, 2001.

[3] N. Boysen, M. Fliedner, and A. Scholl, “A classification of assembly line balancing problems,” *European Journal of Operational Research*, vol. 183, no. 2, pp. 674–693, 2007.

[4] A. Scholl and C. Becker, “State-of-the-art exact and heuristic solution procedures for simple assembly line balancing,” *European Journal of Operational Research*, vol. 168, no. 3, pp. 666–693, 2006.

[5] F. B. Talbot, J. H. Patterson, and W. V. Gehrlein, “A comparative evaluation of heuristic line balancing techniques,” *Management Science*, vol. 32, no. 4, pp. 430–454, 1986.

[6] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* San Francisco, CA: W. H. Freeman, 1979.

[7] T. L. Urban, “Note. optimal balancing of u-shaped assembly lines,” *Management Science*, vol. 44, no. 5, pp. 738–741, 1998.

[8] A. Scholl and R. Klein, “Ulino: Optimally balancing u-shaped jit assembly lines,” *International Journal of Production Research*, vol. 37, no. 4, pp. 721–736, 1999.

[9] I. Baybars, “A survey of exact algorithms for the simple assembly line balancing problem,” *Management Science*, vol. 32, no. 8, pp. 909–932, 1986.

[10] G. R. Aase and N. C. Suresh, “A mathematical programming formulation for u-shaped assembly line balancing,” *International Journal of Production Research*, vol. 41, no. 11, pp. 2545–2569, 2003.

[11] S. Ghosh and C. Gagné, “A comprehensive review of assembly line balancing,” *International Journal of Production Research*, 2011.

[12] Y. Kara, A. İşlier, and Y. Atasagun, “Balancing u-shaped assembly lines: heuristic and metaheuristic approaches,” *Applied Mathematical Modelling*, 2010.

[13] E. Erel and S. Sarin, "A survey of the assembly line balancing problem," *Production Planning & Control*, 1998.

[14] T. Blickle and L. Thiele, "A comparison of selection schemes used in genetic algorithms," in *EVO Conference Proceedings*, 1996.

[15] Y. Park, Y. Kim, and S. Lee, "Precedence preserving genetic operators for scheduling," *Computers & Operations Research*, 2003.

[16] C. Reeves, "Genetic algorithms for the operations researcher," INFORMS, 1993.

[17] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley, 1989.

[18] C. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity.* Dover, 1998.

[19] United Nations General Assembly, "Transforming our world: the 2030 agenda for sustainable development," Resolution A/RES/70/1, 2015, adopted 25 September 2015.

A Python Source Code

Listing 1: Simple Python Example

```
def add(a, b):
    return a + b

print(add(3, 5))
```