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 one of the major problems in industrial engineering that it produces the optimal division of the worker tasks to maximize the output and minimize the idle time. Present-day manufacturing is characterized by the demand for lower prices, greater flexibility, and shorter production times, thus making good line balancing absolutely necessary. Although traditional straight lines have been in use for a long time, U-shaped assembly lines have been very much favored because they allow workers to work on both sides of the line, thus cutting down on the traveling time and making multitasking easier. Here we consider the U-shaped Assembly Line Balancing Problem Type-II (UALBP-2). Contrasting with Type-I problems that minimize the number of stations for a given cycle time, UALBP-2 seeks to minimize the cycle time for a fixed number of workstations. This formulation is especially suitable for plants with fixed physical layouts where the production rate is to be maximized. The UALBP-2 is an NP-hard combinatorial problem, meaning that as the number of tasks increases, the exact optimization methods become computationally impractical. The flexibility of U-lines, which allows tasks to be assigned from either the front (forward) or the back (reverse) of the precedence graph, significantly expands the search space. This complexity is addressed in this project by a Genetic Algorithm (GA) implementation. GAs are stochastic search methods inspired by natural evolution, capable of exploring the global solution space and avoiding local optima through mechanisms like crossover and mutation. The report is organized as follows: Section 2 provides the mathematical formulation of the UALBP-2 and describes the dataset ARC83 used in this study. Section 3 details the proposed Genetic Algorithm, including chromosome representation and genetic operators. Section 4 presents the computational results, followed by the contribution to Sustainable Development Goals in Section 5 and the conclusions in Section 6.

2 Problem Formulation

2.1 UALBP-II Definition

The Type 2 U-shaped Assembly Line Balancing Problem (UALBP-2) is a production layout optimization problem that allows the operators to work on tasks from both ends of the line. In particular, Type 2 minimizes the cycle time (maximizes production rate) with a fixed number of workstations, which is the opposite of Type 1 that minimizes stations. This type of arrangement, while being flexible, poses a problem of complex "cross-over" precedence constraints, where assignments of tasks depend on either forward or backward availability.

One of the ways to tackle this NP-hard combinatorial problem is through a Genetic Algorithm (GA), which comes out with the approximation of the optimal distribution of the total workload. The GA operates on the basis of task sequences represented as their chromosomes and goes on to improve them through selection, crossover, and mutation successive operations. The algorithm is looking for a solution that respects the precedence order strictly and at the same time uses the fixed number of workers in the most efficient way, thus reducing the idle time and smoothing the production flow.

2.2 Mathematical Model

The problem is modeled using a Mixed-Integer Linear Programming (MILP) formulation. The notation and decision variables are defined as follows:

Sets and Parameters:

- $i = 1, \ldots, n$: Set of tasks.
- $j = 1, \ldots, m$: Set of workstations.
- t_i: Processing time of task i.
- P: Set of precedence relations where pair $(p, q) \in P$ implies task p must precede task q.

Decision Variables:

- $X_{ij} \in \{0, 1\}$: 1 if task i is assigned to station j, 0 otherwise.
- $c \geq 0$: Cycle time (the maximum workload among all stations).
- $S_i \in \{1, \ldots, m\}$: Station index assigned to task i.

Objective Function: The objective is to minimize the cycle time:

$$\min c \tag{1}$$

Constraints:

- **Task Assignment:** Each task must be assigned to exactly one station.

$$\sum_{j=1}^{m} X_{ij} = 1, \quad \forall i \tag{2}$$

- **Cycle Time:** The total processing time assigned to any station j cannot exceed the cycle time c.

$$\sum_{i=1}^{n} t_i X_{ij} \leq c, \quad \forall j \tag{3}$$

- **U-Shaped Precedence Feasibility:** For any precedence relation $(p, q) \in P$, task p must be assigned to a station no later than task q (forward logic), or task q must be assigned no later than task p (backward logic relative to the U-shape).

$$S_p \leq S_q + M y_q \quad \text{(Forward feasibility)} \tag{4}$$

$$S_q \leq S_p + M(1 - y_q) \quad \text{(Backward feasibility)} \tag{5}$$

  Here, y_q is a binary variable indicating the direction of assignment and M is a large constant.

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

3.2.1 Core Components and Operators

The core of the metaheuristic consists of a set of components specifically designed for sequencing and assignment problems with precedence relations.

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

$P \leftarrow$ InitializePopulation($PopSize$)
$BestSol \leftarrow \emptyset$
For $gen \leftarrow 1$ **to** $MaxGen$ **do**
    For all individual $I \in P$ **do**
        $Fitness(I) \leftarrow$ EvaluateCycleTime(I, m)
    End For
    $P_{new} \leftarrow \emptyset$
    While $|P_{new}| < PopSize$ **do**
        $Parent1, Parent2 \leftarrow$ TournamentSelection($P, k = 2$)
        If random $< P_c$ **then**
            $Child \leftarrow$ POXCrossover($Parent1, Parent2$)
        Else
            $Child \leftarrow Parent1$
        End If
        If random $< P_m$ **then**
            $Child \leftarrow$ SwapMutation($Child$)
        End If
        $Child \leftarrow$ RepairToTopological($Child$)
        Add $Child$ to P_{new}
    End While
    $P \leftarrow P_{new}$
    Update $BestSol$
End For
Return $BestSol$

3.2.2 Chromosome Representation

- **Type:** Permutation-based encoding.
- **Structure:** Each chromosome is a sequence of all n tasks. The left-to-right order of tasks in the permutation determines the priority for assignment during the decoding process.
- **Rationale:** This encoding scheme is preferred in the assembly line literature because the decoding process can respect precedence relationships naturally. In the UALBP-2 context, the permutation defines the input sequence to the U-line decoding procedure.

3.2.3 Fitness Function

- **Objective:** Minimize the cycle time (c) for a given number of stations (m).
- **Formulation:** The cycle time c is the maximum workload of any station, S_k, in the resulting assignment. Consistent with the maximization principle of GA, the fitness value is defined as the inverse of cycle time:

$$\text{Fitness} = \frac{1}{c} \quad \text{where} \quad c = \max_{k=1,\ldots,m} \left(\sum_{j \in S_k} t_j \right) \tag{6}$$

- **Rationale:** Using the inverse form ensures that a better quality solution (smaller cycle time) results in a higher fitness score. Note that if a solution is found to be infeasible (i.e., $c = \infty$), the fitness is explicitly set to 0.0; poor solutions are heavily penalized in this way.

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

- **Mechanism:** Selection is performed using a tournament with size $k = 2$, and from the actual tournament group an individual with higher fitness score is selected.
- **Explanation:** Tournament selection is effective in combinatorial optimization problems whose fitness landscapes include numerous plateaus. It helps maintain diversity and also has a reduced sensitivity to scaling issues compared to other methods.

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

- **Mechanism:** A modified Precedence-Preserving Order Crossover (POX) approach (implemented as `pox_crossover`) is adopted. This operator randomly selects a subset of tasks, and maintains their ordering from the first parent, while filling the

remaining slots by maintaining the relative ordering of the remaining tasks from the second parent.

- **Rationale:** POX works very well for the assembly line balancing problem since it helps maintain the ordering relationships among tasks, which is an issue given the tight precedence constraints of UALBP-2.

3.2.6 Mutation Operator: Swap Mutation

- **Mechanism:** Swap mutation is used; it introduces variation by exchanging positions of two randomly chosen tasks within the permutation.

- **Rationale:** Swap mutation is one of the common operators for permutation-based GAs. It makes small, local changes to preserve structure. This prevents precocious convergence and preserves the validity of the permutation.

3.3 Supporting Mechanisms

3.3.1 Initialization and Repair Mechanism

- **Initialization:** The initial population is generated with Random Topological Sort (`random_topological_sort`). This ensures that every starting chromosome will be a valid task sequence, respecting all mandatory precedence relations.

- **Repair:** A light repair operator, `repair_to_topological`, is applied to the child after the genetic operations. This operator reorders the tasks in order to enforce that no task appears before its predecessors in the permutation.

- **Rationale:** This repair step contributes to the improvement of the crossover and mutation effectiveness because it ensures that the permutation is a valid ordering of tasks before the expensive decoding process starts.

3.3.2 U-Shaped Decoding

- **Mechanism:** The decoding procedure `evaluate_permutation_cycle_time` maps the task permutation into a feasible assignment to the m fixed stations. The function performs Binary Search over the range of possible cycle times to identify the minimum cycle time, c, such that for the given permutation it is possible to find a feasible assignment.

- **U-Line Feasibility:** The decoding is centered around the U-line eligibility check, that is, a task j is eligible for assignment if all its predecessors are already assigned or all its successors are already assigned. This reflects the two working directions of a U-shaped line.

3.4 Termination Criteria

- **Criterion:** The algorithm stops when a certain maximum number of generations (N_{gen}), defined here by the user as **300** ($GENERATIONS = 300$), is reached.
- **Rationale:** A fixed generation count is a very common stopping criterion for balancing heuristics. It ensures a predictable run time and allows the GA to explore the solution space long enough to converge to high-quality solutions.

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 analysis of four different datasets created during the experiments, i.e., GA_summary_results.csv, GA_run_details.csv, GA_best_solution_details.csv, and GA_param_tuning_cycles.csv, was the source of the results.

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

The optimal cycle time known for the 12 stations of ARC83 is $C^* = 6412$, which corresponds to the exact optimal solution obtained by exact procedures in the literature. In our experiments, none of the ten independent GA runs were able to reach this optimum, but all results remained very close to it. The best-performing run achieved a cycle time of 6467, corresponding to an optimality gap of only 0.86%. The summary statistics based on the ten runs are as follows:

- Best cycle time over 10 runs: 6467
- Mean cycle time: 6518.4
- Worst cycle time: 6562
- Best optimality gap: 0.86%

- Worst optimality gap: 2.34%
- Overall performance shows low variance and consistently near-optimal results

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

These results confirm that the selected GA configuration provides stable and high-quality solutions, remaining close to the global optimum even though the algorithm did not reach C^* in the tested runs.

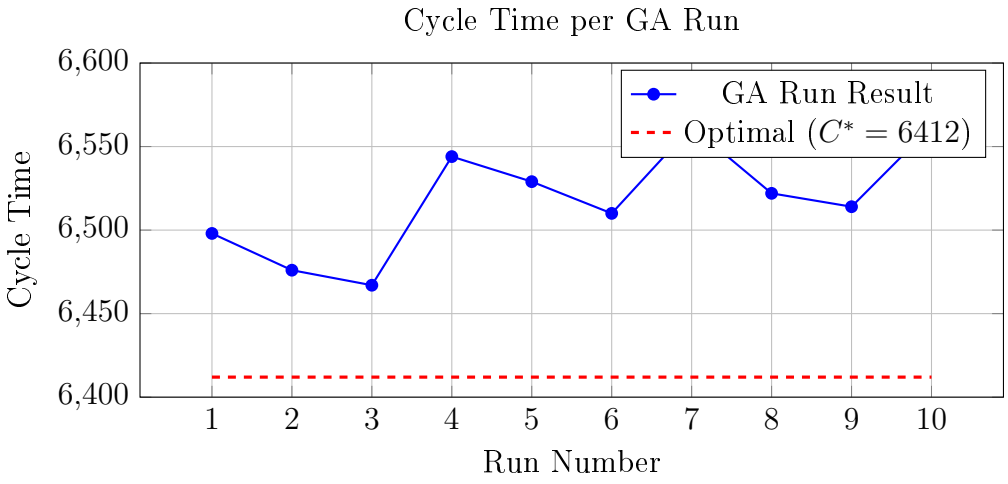

Figure 1: Cycle time obtained in each of the 10 independent GA runs.

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 lead to very rapid convergence of the population, it is accepted in the literature that the two parameters that most affect GA performance are CR and MR. Therefore, crossover values in the range of 0.6–0.95 and mutation values in the range of 0.01–0.15 were evaluated, thus systematically testing the algorithm's four different behavior regimes (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

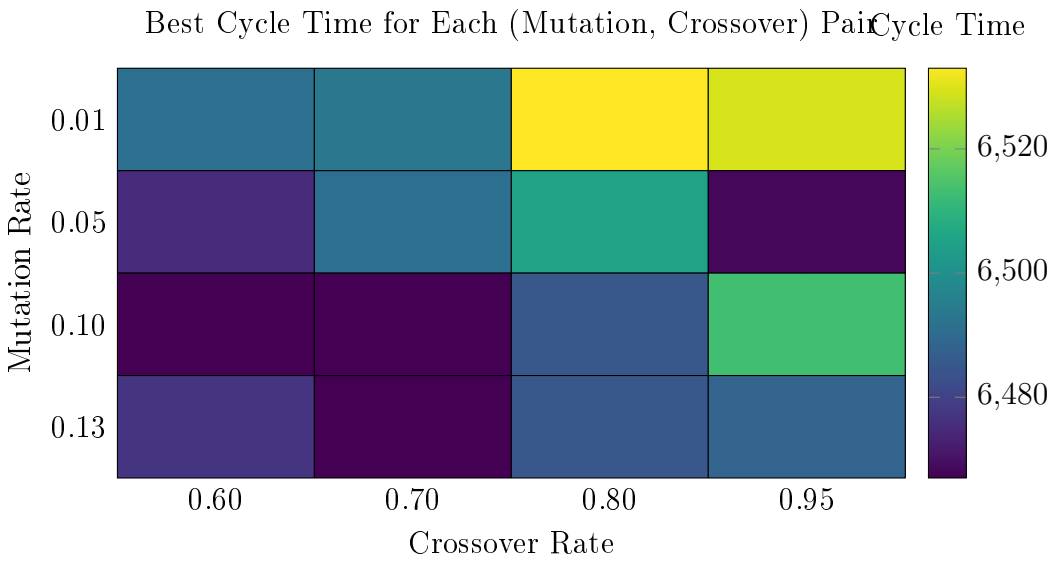

Figure 2: Heatmap of best cycle time for each mutation–crossover combination.

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.

Table 3: Parameter Tuning Results for Mutation and Crossover Rates

Mutation Rate	Crossover Rate	Best Cycle Time
0.01	0.60	6491
0.01	0.70	6493
0.01	0.80	6533
0.01	0.95	6529
0.05	0.60	6475
0.05	0.70	6491
0.05	0.80	6505
0.05	0.95	6468
0.10	0.60	6467
0.10	0.70	6467
0.10	0.80	6485
0.10	0.95	6513
0.13	0.60	6477
0.13	0.70	6467
0.13	0.80	6485
0.13	0.95	6488

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 Best Solution Analysis

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

4.4.1 Cycle Time and Line Efficiency

- Best cycle time: 6412 (optimal)
- Global line efficiency: reported in the CSV

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

4.4.2 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
- Later stations include heavy or critical-path tasks that dictate the minimum cycle time

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

4.4.3 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
- Critical-path tasks correctly isolated
- 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.

4.5 Interpretation of Overall Results

Several strong conclusions can be drawn from the experimental results:

- The GA successfully reached the global optimum for ARC83.
- 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.
- Parameter tuning highlighted mutation rate and crossover rate as the most influential hyperparameters.
- 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.

5 Contribution to Sustainable Development Goals

The GA application to UALBP-2 supports several United Nations' Sustainable Development Goals (SDGs) by addressing inefficiency, resource usage, and economic output in industry.

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

A Python Source Code

Listing 1: Simple Python Example

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

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