systems

MDPI

*Article*

# Type-1 Robotic Assembly Line Balancing Problem That Considers Energy Consumption and Cross-Station Design

**Yuanying Chi [1], Zhaoxuan Qiao [1], Yuchen Li [1,*], Mingyu Li [2] and Yang Zou [2]**

[1] School of Economics and Management, Beijing University of Technology, Beijing 100124, China
[2] Beijing Benz Automotive Co., Ltd., Beijing 100176, China
* Correspondence: liyuchen@bjut.edu.cn

**Abstract:** Robotic assembly lines are widely applied to mass production because of their adaptability and versatility. As we know, using robots will lead to energy-consumption and pollution problems, which has been a hot-button topic in recent years. In this paper, we consider an assembly line balancing problem with minimizing the number of workstations as the primary objective and minimizing energy consumption as the secondary objective. Further, we propose a novel mixed integer linear programming (MILP) model considering a realistic production process design—cross-station task, which is an important contribution of our paper. The "cross-station task" design has already been applied to practice but rarely studied academically in type-1 RALBP. A simulated annealing algorithm is developed, which incorporates a restart mechanism and an improvement strategy. Computational tests demonstrate that the proposed algorithm is superior to two other classic algorithms, which are the particle swarm algorithm and late acceptance hill-climbing algorithm.

**Keywords:** robotic assembly line balancing; energy consumption; cross-station task; mixed integer liner programming; simulated annealing



**Citation:** Chi, Y.; Qiao, Z.; Li, Y.; Li, M.; Zou, Y. Type-1 Robotic Assembly Line Balancing Problem That Considers Energy Consumption and Cross-Station Design. *Systems* **2022**, *10*, 218. https://doi.org/10.3390/systems10060218

Academic Editor: William T. Scherer

Received: 30 September 2022
Accepted: 4 November 2022
Published: 15 November 2022



## 1. Introduction

Assembly lines are widely utilized in mass production, such as automobiles and household appliances. Robots have gained appeal with line designers as a result of advances in manufacturing technology and the quest for production efficiency. Assembly line with robots is referred to as the robotic assembly line. Compared with human workers, robots can process tasks quickly and accurately without having to worry about undue fatigue. Further, a robot with multiple arms is also more adaptable and capable of processing a variety of tasks [1]. However, the utilization of robots will consume energy and create pollution problems. Fysikopoulos et al. [2] pointed out that approximately 9–12% of the cost of manufacturing a car is spent on energy, and for every 20% reduction in energy consumption, the final manufacturing cost will drop by roughly 2–2.4%. Therefore, reducing energy consumption plays a vital role in protecting the environment and boosting business competitiveness.

The assembly line balancing problem (ALBP) was first raised by Bryton [3], and refers to assigning tasks to workstations to achieve the optimization objective. The ALBP can be divided into four types according to different inputs and objectives, i.e., minimizing the number of workstations with a given cycle time is called type-1 ALBP. Minimizing the cycle time given the number of workstations is called type-2 ALBP. If the number of workstations and cycle time are unknown simultaneously, maximizing the line efficiency is labeled type-E ALBP. Aiming to find a feasible balancing solution when both are given is type-F ALBP.

The ALBP with robots is called the robotic assembly line balancing problem (RALBP), which was first proposed by Rubinovitz and Bukchin [4], and is a significant problem in the industrial sector. Since different types of robots have different processing task times and

energy consumption per unit of time, the allocation of robots also needs to be considered in RALBP. In this paper, we consider minimizing the number of workstations with a given cycle time, i.e., type-1 RALBP, to be the primary objective and minimizing energy consumption based on the optimal number of workstations to be the secondary objective. Further, we introduce the "cross-station task" design, which is quite popular in practice but is rarely considered in academic settings [5], to improve production efficiency in the RALBP, and propose a novel mixed integer linear mathematical model for this problem. A simulated annealing algorithm encapsulating a restart mechanism and an improvement rule are developed to solve the problem.

The type-1 ALBP has been widely studied in the literature. Kilincci [6] presented a heuristic algorithm based on the Petri net approach to solve the type-1 simple ALBP. Manavizadeh et al. [7] solved a U-shaped balancing type-1 problem with different types of workers and designed an alert system based on the optimal number of stations to balance workload. A simulated annealing algorithm (SA) was used to solve this problem. Li et al. [8] investigated 14 meta-heuristics for type-1 two-sided ALBP and two new decoding schemes with a reduced search space developed. Comprehensive experiments have shown that the improved iterated greedy algorithm is the most efficient in solving the benchmark problems. Li et al. [9] investigated type-1 assembly line balancing considering uncertain task time. An algorithm based on the branch and bound remember algorithm was developed to solve this problem, and the effectiveness of the algorithm was demonstrated. Li et al. [10] incorporated uncertain task time attributes in type-1 U-shaped ALBP. They proposed an algorithm based on the branch and bound remember algorithm to solve this problem. Zhang [11] proposed an immune genetic algorithm (IGA) which aimed to minimize the number of workstations as well as the workload. Baskar and Anthony Xavior [12] investigated a few heuristic algorithms based on slope indices, which is a method of assigning tasks to stations, to solve the type-1 simple ALBP. Pınarbaşı and Alakaş [13] formulated a constraint programming (CP) model for type-1 ALBP considering assignment restrictions. The author compared the results of different models and showed that CP is the best one. Huang et al. [14] considered a mixed-model two-sided ALBP that aimed at minimizing the number of mated-stations. A combinatorial Benders-decomposition-based exact algorithm was used to solve the proposed problem. The computational tests showed that this algorithm can obtain exact results on large-sized problem instances.

Ever since Rubinovitz and Bukchin [4] came up with RALBP and proposed an efficient heuristic to solve it [15], it has become a prevalent research direction for ALBP. Hong and Cho [16] solved the type-1 RALBP considering assembly cost. A simulated annealing algorithm was utilized as the optimization tool. Gao et al. [17] proposed an innovative genetic algorithm (GA) hybridized with local search to solve the type-2 RALBP. Five local search procedures were developed to enhance the search ability of GA. Li et al. [1] designed a cuckoo search method through different neighborhood generation methods to solve the two-sided RALBP. The computational tests showed that the proposed algorithm outperformed other meta-heuristics. Janardhanan et al. [18] considered sequence-dependent setup times for RALBP to minimize the cycle time. They proposed a migrating birds optimization algorithm (MBO) and demonstrated the effectiveness of the MBO. Sun and Wang [19] developed a hybrid algorithm that combines the branch-and-bound (B&B) and estimation of the distribution algorithm to minimize the cycle time on the robotic assembly line. Aslan [20] investigated an two-sided RALBP with sequence-dependent setup times, and a variable neighborhood search (VNS) algorithm was utilized to solve this problem.

Other features of RALBP have also been studied. Michels et al. [21] studied spot welding robotic assembly lines based on an automotive company located in Brazil. A mixed-integer linear programming (MILP) model was developed. Pereira et al. [22] solved cost-oriented RALBP (cRALBP), taking into account that different types of robots have different costs. Rabbani et al. [23] investigated four-sided human–robot collaborations on ALBP, where the tasks are performed on the left, right, above, and beneath sides. An augmented multi-objective particle swarm optimization was used to solve the model. Koltai et al. [24]

analyzed the short- and long-term effects of adding robots to human ALs for the operation of the line. The use of the models was demonstrated using a case study involving a power inverter. Lahrichi et al. [25] investigated two variants of type-2 RALBP with sequence dependence. The first variant is given different types of robots, each of which can be arbitrarily assigned to multiple stations. Another variant is that given a group of robots, each robot can only be assigned to one station.

Further, the use of robots on assembly line creates energy consumption problems; some scholars treat energy consumption as one of the optimization objectives. Mukund Nilakantan et al. [26] proposed models with dual objectives to minimize the cycle time and total energy consumption simultaneously. The particle swarm optimization was used to solve this problem. Nilakantan et al. [27] proposed a multi-objective co-evolutionary algorithm to solve the energy-related RALBP. Zhang et al. [28] investigated a U-shaped RALBP and developed a multiobjective mixed-integer non-linear model to optimize carbon emissions. Hybrid Pareto–grey wolf optimization (HPGWO) was designed, and its effectiveness was demonstrated. Zhou and Wu [29] aimed to optimize the total energy consumption and a productivity-related objective simultaneously in RALBP. A novel algorithm based on a well-known enhanced decomposition-based multi-objective algorithm (MOEA/D) was designed to solve this problem. Zhang et al. [30] investigated mixed-model U-shaped RALBP and proposed a hybrid multi-objective dragonfly algorithm (HMODA) to achieve the goals of energy saving and efficiency. Belkharroubi and Yahyaoui [31] minimized energy consumption on a mixed-model RALBP. A cuckoo search algorithm, which was based on the memory principle, was developed to tackle this problem. The authors tested its effectiveness by comparing other algorithms.

By analyzing the previous research works, we can conclude that the RALBP had been studied extensively from various angles. However, the mathematical models are always nonlinear, which is not desired for a commercial solver specialized in solving mixed-integer problems, e.g., CPLEX. We did not find any literature about the type-1 RALBP with the energy consumption objective. Further, the multi-functional robots and their implied application to the assembly line regarding the cross-station design are absent.

The contribution of our paper is as follows: (1) To our best knowledge, we did not find the studies of the multi-objective optimized type-1 RALBP considering energy consumption. Thus, this work fills the research gap in RALBP. (2) We introduce the "cross-station task" design, which has already been applied in practice but rarely studied academically, into type-1 RALBP for the first time. (3) We leverage and modify the simulated annealing algorithm for solving this problem, where incorporates an improvement mechanism of exact algorithms.

The remainder of this paper is organized as follows. Section 2 describes the cross-station task design in detail, and a simple example is presented. We propose a MILP model considering the cross-station task design in Section 3. A simulated annealing algorithm is designed for the problem in Section 4. Computational results are shown in Section 5. Section 6 concludes the paper.

## 2. RALBP-CS Design

In this section, we introduce the cross-station task design to reduce the idle time of each station as much as possible on the assembly line. This idea is similar to certain other studies. Grzechca and Foulds [32] relaxed the assumption that a task cannot be split among two or more stations, i.e., a task can be split into multiple subtasks, then changed the priority graph for research. Nanda and Scher [33] relaxed the assumption and studied overlapping workstations, where a task can be processed by a pair of workstations simultaneously.

In the cross-station task design, a task could be processed at three stations, which is a more realistic design than the previous two designs in some manufacturing systems. The cross-station task design is a practical application [5]. The three stations include the current station and its front and rear stations, if they exist. To achieve this, one station can "borrow" ("lend") its cycle time from (to) its front or rear stations. If this task is assigned

to the very front part of the current station, it could be processed in advance at the front station. That is, the current station borrows its cycle time from the front station. If this task is assigned to the very rear part of the current station, it could be processed with a delay at the rear station. That is, the current station borrows its cycle time from the rear station. For example, in Figure 1, task 2 is originally assigned to station 2. Since station 2 has borrowed a portion of the cycle time of station 1, task 2 can start being processed in advance at station 1. Likewise, task 4 is originally assigned to station 3. Since station 2 has lent a portion of its cycle time to station 3, task 4 can start being processed in advance at station 2. There are two points worth noting: (1) Tasks can only be assigned to one station and one robot, but they can be processed when the work-in-process (WIP) is at adjacent stations. (2) A robot is multi-functional with different robotic arms, as mentioned in Section 1. For example, robot 3 is processing task 4 with one robotic arm and operating task 6 with another arm.

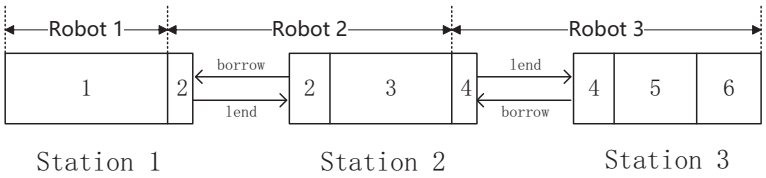


**Figure 1.** Assembly line with cross-station task design.

Then, a simple example is presented.

**Example 1.** *To intuitively recognize the advantages of this design, suppose there is only one task sequence and a type of robot, which is shown in Figure 2. The task number is stored in the node, and the number outside the code displays the time the robot takes to process the task. There is a given cycle time of 11.*

*Task assignment is shown in Figure 3. The shaded part represents idle time. Without the RALBP-CS design, tasks 1, 2, and 3 are assigned to station 1; tasks 4, 5 and 6 are assigned to station 2; and tasks 7, and 8 are assigned to station 3. Thus, there are 3 stations installed, and some idle time is incurred on the line. If the RALBP-CS design is applied to the line, tasks 1, 2, 3, and 4 are assigned to station 1, which borrows 1 unit of the cycle time of station 2. Then, tasks 5, 6, 7, and 8 are assigned to station 2. The number of stations that are installed is two (one less than the line without considering RALBP-CS design), and there is no idle time at all on line.*

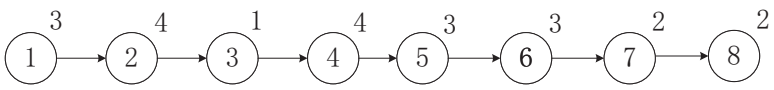


**Figure 2.** Priority relationship for example 1.

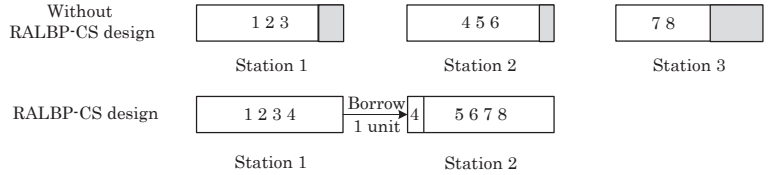


**Figure 3.** Task assignment for example 1.

## 3. Mathematical Modeling

In this section, the MILP mathematical model is formulated. The primary goal is to minimize the number of workstations. The secondary objective is to minimize total energy consumption. The decision variables are summarized at Table 1.

**Table 1.** Notations.

| Parameters | Descriptions |
|---|---|
| $n$ | Total number of tasks |
| $R$ | Total number of robot types |
| $J$ | The maximum number of workstations that are allowed to be opened |
| $m$ | The number of workstations that are actually installed |
| $i, h$ | Index of tasks, $i, h = 1, 2, \cdots n$ |
| $s, j$ | Index of stations. $s, j = 1, 2, \cdots J$ |
| $r$ | Index of robots. $r = 1, 2, \cdots R$ |
| $c$ | Cycle time |
| $\omega$ | Weight coefficient |
| $Rte_s$ | The remaining capacity at station $s$ |
| $TEC$ | Total energy consumption |
| $TEC_{max}$ | The upper bound of total energy consumption |
| $OEC_r$ | Operation energy consumption of the robot $r$ per time unit |
| $SEC_r$ | Standby energy consumption of the robot $r$ per time unit |
| $t_{ir}$ | The task $i's$ processing time by robot $r$ |
| $Pr(i)$ | Set of direct predecessors of task $i$ |
| $\gamma$ | Maximum amount of time that can be borrowed from one station by another |
| $\phi$ | A large positive number |
| **Decision variables** | **Descriptions** |
| $x_{ij}$ | 1, if task $i$ is assigned to workstation $j$ ; 0, otherwise |
| $y_{rj}$ | 1, if robot $r$ is allocated to workstation $j$ ; 0, otherwise |
| $z_{irj}$ | 1, if task $i$ and robot $r$ are assigned to workstation $j$ ; 0, otherwise |
| $d_{j,s}$ | 1, if station $j$ borrows time from station $s$; 0, otherwise |
| $q_{j,s}$ | A positive value shows the amount of time station $j$ borrows from station $s$; 0, otherwise |

### *3.1. Assumptions*

The basic assumptions underlying the problem are as follows.

- One single product is manufactured on the assembly line.
- Robots are multi-functional with numerous arms that can handle different tasks simultaneously.
- The precedence relations between the tasks are given previously.
- A task can only be assigned to one station and one robot.
- Each station can only borrow time from its adjacent stations.
- The task-processing time is dependent on the type of robot assigned to it.
- Each robot can be assigned to any station and can process any task.

The notations are presented in Table 1 and used throughout the paper.

### *3.2. Formulation*

In the model, $TEC$ represents the total energy consumption. $TEC_{max}$ represents the upper bound of total energy consumption, which is a fixed parameter by taking the maximum operating and idle energy consumption of the robots for each task, and then summing them. Thus, $TEC/TEC_{max}$ is in the range [0, 1]. Now, the mathematical model is presented.

$$\min \quad Obj = m + TEC/TEC_{max}, \tag{1}$$

$$s.t. \quad m = \sum_{r=1}^{R}\sum_{j=1}^{J} y_{rj}, \tag{2}$$

$$TEC = OEC + SEC, \tag{3}$$

$$OEC = \sum_{j=1}^{J}\sum_{r=1}^{R}\sum_{i=1}^{n} OPC_r z_{irj} t_{ir}, \tag{4}$$

$$SEC = \sum_{j=1}^{J}\sum_{r=1}^{R} g_{rj} SPC_r, \tag{5}$$

$$g_{rj} \le c + q_{j,j+1} + q_{j,j-1} - q_{j+1,j} - q_{j-1,j} - \sum_{r=1}^{R}\sum_{i=1}^{n} z_{irj} t_{ir} + (1 - y_{rj})\psi, \forall j = 1, \cdots, J, \tag{6}$$

$$g_{rj} \ge c + q_{j,j+1} + q_{j,j-1} - q_{j+1,j} - q_{j-1,j} - \sum_{r=1}^{R}\sum_{i=1}^{n} z_{irj} t_{ir} - (1 - y_{rj})\psi, \forall j = 1, \cdots, J, \tag{7}$$

$$\sum_{r=1}^{R} y_{rj} \le 1, \forall j = 1, \cdots, J, \tag{8}$$

$$\sum_{r=1}^{R} y_{rj} \ge x_{ij}, \forall j = 1, \cdots, J, \forall i = 1, \cdots, n, \tag{9}$$

$$\sum_{r=1}^{R} y_{rj} \le \sum_{i=1}^{n} x_{ij}, \forall j = 1, \cdots, J, \tag{10}$$

$$\sum_{r=1}^{R} y_{rj} \ge \sum_{r=1}^{R} y_{rj+1}, \forall j = 1, \cdots, J-1, \tag{11}$$

$$\sum_{j=1}^{J} x_{ij} = 1, \forall i = 1, \cdots, n, \tag{12}$$

$$\sum_{i=1}^{n} j x_{hj} \le \sum_{i=1}^{n} j x_{ij}, \forall h \in p(i), \tag{13}$$

$$x_{ij} + y_{rj} \le z_{irj} + 1, \forall i = 1, \cdots, n, \forall r = 1, \cdots, R, \forall j = 1, \cdots, J, \tag{14}$$

$$\left(1 - x_{ij}\right) + y_{rj} \le \left(1 - z_{irj}\right) + 1, \forall i = 1, \cdots, n, \forall r = 1, \cdots, R, \forall j = 1, \cdots, J, \tag{15}$$

$$x_{ij} + \left(1 - y_{rj}\right) \le \left(1 - z_{irj}\right) + 1, \forall i = 1, \cdots, n, \forall r = 1, \cdots, R, \forall j = 1, \cdots, J, \tag{16}$$

$$\left(1 - x_{ij}\right) + \left(1 - y_{rj}\right) \le \left(1 - z_{irj}\right) + 1, \forall i = 1, \cdots, n, \forall r = 1, \cdots, R, \forall j = 1, \cdots, J, \tag{17}$$

$$q_{j,s} \le \gamma, \forall j = 1, \cdots, J, \forall s = 1, \cdots, J, \tag{18}$$

$$d_{j,j+1} + d_{j+1,j} \le 1, \forall j = 1, \cdots, J-1, \tag{19}$$

$$\varphi d_{j,j+1} \ge q_{j,j+1}, \forall j = 1, \cdots, J-1, \tag{20}$$

$$\varphi d_{j+1,j} \ge q_{j+1,j}, \forall j = 1, \cdots, J-1, \tag{21}$$

$$q_{j+1,j} \ge 0.1 d_{j+1,j}, \forall j = 1, \cdots, J-1, \tag{22}$$

$$q_{j,j+1} \ge 0.1 d_{j,j+1}, \forall j = 1, \cdots, J-1, \tag{23}$$

$$\sum_{r=1}^{R} y_{rj+1} \ge d_{j,j+1}, \forall j = 1, \cdots, J-1, \tag{24}$$

$$\sum_{r=1}^{R} y_{rj+1} \ge d_{j+1,j}, \forall j = 1, \cdots, J-1, \tag{25}$$

$$q_{1,0}, q_{0,1}, q_{J,J+1}, q_{J+1,J} = 0, \tag{26}$$

$$q_{j,s} \ge 0, \forall j = 1, \cdots, J, \forall s = 1, \cdots, J, \tag{27}$$

$$g_{rj} \ge 0, \forall r = 1, \cdots, R, \forall j = 1, \cdots, J, \tag{28}$$

Equation (1) shows the objective function, which represented by the primary ($m$) and secondary objective ($TEC$). Constraint (2) calculates the number of workstations that are actually installed. Constraint (3) calculates the total energy consumption by summing the total operation energy consumption (Constraint (4)) and standby energy consumption (Constraint (5)) on the assembly line. Constraints (6)– (7) calculate the idle time for each station, and if the robot $r$ is allocated to the station $j$ ($y_{rj} = 1$), the constraints are active. Constraint (8) refers to the requirement that, at most, one robot can be allocated to the station. Constraint (9) ensures that tasks are assigned only to stations that are installed. Constraint (10) ensures that the installed station is not empty. Constraint (11) ensures the stations are installed continuously in turn. Constraint (12) ensures a task can only be assigned to one station. Constraint (13) refers to the priority constraint. Constraints (14)–(17) demonstrate the logical relationship between tasks, robots, and workstations. Constraint (18) limits the upper bound of time borrowing from one station to another. Constraints (19)–(23) ensure that two stations cannot borrow each other's time simultaneously. Constraints (24)–(25) ensure that the action of lending and borrowing cannot occur for the empty station. Constraints (26)–(28) are the domain constraints.

## 4. A Simulated Annealing Algorithm

In this section, a simulated annealing algorithm (SA) is designed to tackle the proposed problem. SA was first proposed by Kirkpatrick et al. [34], which is an optimization algorithm that imitates the gradual cooling of metals. It is a meta-heuristic including a random optimization approach, which is to avoid a local optimum by evaluating inferior solutions. It has the advantages of simple description, flexible use, wide application, high operation efficiency, and less affected by the input parameter. Starting from a randomly chosen initial solution $S$, SA seeks a candidate solution $S'$ in the area surrounding the initial solution. $S$ and $S'$ correspond to fitness values $Obj$ and $Obj'$, respectively. Then, determine which candidate solution can be approved by comparing the fitness values of the candidate with the present solutions. The amount of change in the fitness value is referred to as $\Delta$ ($\Delta = Obj - Obj'$). If $\Delta > 0$, $S'$ is accepted; otherwise, $S'$ is accepted with a given probability ($p = e^{-\frac{\Delta}{T}}$), where $T$ is the temperature parameter. At the beginning, there is a high probability of accepting the inferior solution due to the higher $T$. In each iteration, $T$ is decreased by a cooling schedule until a predetermined stopping requirement is satisfied.

### *4.1. The General Framework of SA*

In this study, the optimal objective is solved by designing the iterative mechanism and developing an improvement rule based on the traditional SA. $Obj$ represents the fitness value. The notation of SA is given below.

| | |
|---|---|
| $T$ | Temperature parameter |
| $\alpha$ | Cooling rate |
| $ite$ | The iteration index |
| $ite_{max}$ | The maximum number of iterations of temperature |
| $dn_{max}$ | The maximum number of iterations per restart |
| $Sq_{task}$ | A feasible assignment sequences of task |
| $Sq_{robot}$ | A feasible assignment sequences of robot |
| $rn$, $pr$ | Uniform random numbers between [0, 1] |

Specifically, SA searches for the $Sq_{task}$ and $Sq_{robot}$ that result in the best fitness. Variable neighborhood search (VNS) is another characteristic of the SA. In SA, VNS chooses between two neighborhoods and systematically searches them. A random number ($pr$) determines whether to alter the robot sequence or the task sequence for each iteration. The iterative steps of SA are listed below.

Step 1: Input $P(i)$, $T$, $Nn$, $\alpha$, $OPC_r$, $SPC_r$, $t_{ir}$, $J$, $c$, $dn_{max}$, $TEC_{max}$, set $ite = 1$, $dn = 0$.

Step 2: Initialize sequences $Sq_{task}$ and $Sq_{robot}$.

Step 3: Generate the initial objective value $Obj$. Set the first three optimal objective values equal to $Obj$.

Step 4: Generate $pr$, if $pr \leq 0.5$, launch the neighborhood generation mechanism of the task sequence to obtain a new $Sq'_{task}$; otherwise, launch the neighborhood generation mechanism of the robot sequence, and obtain a new $Sq'_{robot}$.

Step 5: Launch decoding process to obtain a new objective value $Obj'$.

Step 6: Calculate $\Delta = Obj' - Obj$. If $\Delta < 0$, update and retain the first three optimal solutions, update $dn = 0$, go to step 8; otherwise, update $dn = dn + 1$, and go to step 7.

Step 7: Generate $rn$, if $rn < e^{-\frac{\Delta}{T}}$, go to step 8; otherwise, go to step 9.

Step 8: Accept and update $Sq_{task} = Sq'_{task}$, $Sq_{robot} = Sq'_{robot}$, $Obj = Obj'$.

Step 9: If $dn == dn_{max}$, activate the restart mechanism, update $dn = 0$, go to step 2; otherwise, update $T = \alpha T$, $ite = ite + 1$, and go to step 10.

Step 10: If $ite == ite_{max}$, launch the improvement mechanism, output optimal solution, done.

### *4.2. Initial Sequence Encoding*

In the initial sequence encoding process, we employ $Sq_{task}$ and $Sq_{robot}$ to express the feasible assignment sequences of task and robot, respectively. In contrast to the task sequence, a feasible robot sequence can be generated randomly because the robot sequence is not constrained by precedence. The encoding details are provided below for generating $Sq_{task}$ and $Sq_{robot}$.

Step 1: Generate $Sq_{robot}$, which is an array containing $J$ random integers taken from 1 to $R$.

Step 2: Set $Sq_{task} = [\quad]$.

Step 3: Generate $Sq'_{task}$, which is an array containing random permutation of the integers from 1 to $n$.

Step 4: Based on the $Sq'_{task}$ and the precedence constraints, assign the task $i$ to $Sq_{task} = [\quad]$, then delete the task $i$ in $Sq'_{task}$.

Step 5: If task $i$ cannot be assigned because of violating precedence relationship, skip it and then consider the next task $i$ according to $Sq'_{task}$, then go to step 4.

Step 6: Repeat steps 4–5 until $Sq'_{task} = [\quad]$, obtain $Sq_{task}$, done.

### *4.3. Decoding of Objective Function*

In the decoding process, each robot is allocated to a station in turn according to $Sq_{robot}$. The robot allocation at stations $s$ and $s+1$, respectively, are denoted by $r$ and $r'$. The initialization process (step 1) refers to loading the input data. The task assignment process is described in steps 2 to 5, where step 2 refers to assigning the task to the current station, and steps 3–5 refer to assigning the task to the current station or next adjacent station considering the cross-station design. These three points should be noted: (1) if the final task needs to borrow time of the assignment process, it is assigned to the next adjacent station directly to prevent the situation that an empty stations lend its time; (2) the current station's available time is not $Ret_s$ but $Ret_s + \gamma$ since the station is permitted to lend or borrow time; and (3) if the task is assigned to station $s+1$, the idle time $\max(Ret_s - \gamma, 0)$ may incur at station $s$, i.e., $\min(Ret_s, \gamma)$ is the amount of time that station $s$ lends to station $s+1$. Step 6 computes the energy consumption for station $s$. Eventually, the feasible solutions $Obj$ can be obtained in step 7.

Step 1: Input $t_{ir}$, $J$, $Sq_{task}$, $Sq_{robot}$, $OPC_r$, $SPC_r$, set $s = 1$, $id = 1$, and $Ret_s = c$.

Step 2: If all tasks are assigned, go to step 7; otherwise, assign the $id_{th}$ task in $Sq_{task}$ to station $s$, update $Ret_s$ ($Ret_s = Ret_s - t_{ir}$) and $id = id + 1$, and go to step 3.

Step 3: If $t_{ir} > Ret_s$ and $id == n$, assign the task $i = Sq_{task}(id)$ to station $s+1$, set $q_{s,s+1} = 0$ and $q_{s+1,s} = 0$, update $s = s + 1$, and compute $OEC_s$ and $SEC_s$, go to step 7; elseif $t_{ir} > Ret_s$ and $id < n$, go to step 4; otherwise, go to step 3.

Step 4: If $t_{ir} \leq \gamma + Ret_s$ and $t_{ir'} + \max(Ret_s - \gamma, 0) \geq t_{ir}$, assign the task $i = Sq_{task}(id)$ to station $s$, and set $q_{s,s+1} = t_{ir} - Ret_s$, $q_{s+1,s} = 0$, update $Ret_{s+1}$, where $Ret_{s+1} = c - q_{s,s+1}$, go to step 6; otherwise, go to step 5.

Step 5: Assign the task $i = Sq_{task}(id)$ to station $s+1$, set $q_{s,s+1} = 0$ and $q_{s+1,s} = \min(Ret_s, \gamma)$, update $Ret_{s+1}$, where $Ret_{s+1} = c + q_{s+1,s} - t_{ir}$, and go to step 6.

Step 6: Compute $OEC_s$ and $SEC_s$, update $id = id + 1$, and $s = s + 1$, go to step 2.

Step 7: Get $m$, where $m = s$, compute $TCF$, and output $Obj$.

The decoding operation of objective function is then illustrated with an example.

**Example 2.** *Consider the precedence relationship graph given in Figure 4 and the parameters are provided in Table 2.* $Sq_{task} = \{1, 2, 4, 3, 5, 6, 7, 8\}$, $Sq_{robot} = \{3, 2, 2, 1, 3\}$. *Tasks 1 and 2 are assigned to station 1 and update* $Ret_1$, $Ret_1 = 3$. *Then we should assign task 4 according to* $Sq_{task}$ *since* $t_{4,3} > Ret_1$ *and task 4 is not the last. We should take into account the cross-task design. By executing step 4,* $4 < 3 + 2$ *and* $3 + \max(3 - 2, 0) = 4$, *task 4 should be assigned to station 1 and station 1 shall borrow 1 unit of time from station 2. Update* $Ret_2$, $Ret_2 = 10$, *tasks 3 and 5 are assigned to station 2, and update* $Ret_2$, $Ret_2 = 1$. *Task 6 is assigned to station 3 due to* $4 > 1 + 2$, *and station 2 lends 1 unit of time to station 3. Task 7 is assigned to station 3 and we update* $Ret_3$, $Ret_3 = 3$. *When assigning task 8, we find* $t_{8,2} > Ret_3$ *and task 8 is the last, and thus task 8 is assigned to station 4 directly. Finally, a feasible solution can be found, where* $m = 4$, $TCF = 9.825$, *and* $Obj = 4.856$.

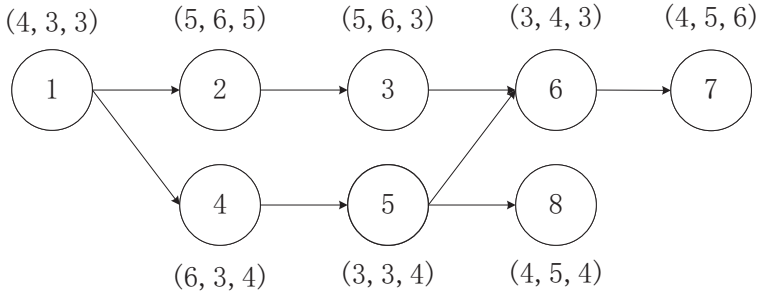


**Figure 4.** Priority relationship for example 2.

**Table 2.** Parameters for example 2.

| Variables and Parameters | Value |
|---|---|
| $n$ | 8 |
| $R$ | 3 |
| $J$ | 5 |
| $c$ | 11 |
| $\gamma$ | 2 |
| $TEC_{max}$ | 11.48 |
| $OEC_r$ | [0.3; 0.25; 0.32] |
| $SEC_r$ | [0.03; 0.025; 0.032] |

### *4.4. Neighborhood Generation and Restart Mechanism*

In SA, an insert method mentioned by Khorasanian et al. [35] is used for generating a neighbor of the task sequence. Simply put, a new task sequence $Sq_{task}$ can be generated by relocating a task to a different position. The reader can refer to the cited literature for more details on the neighborhood generation of task sequences. For neighborhood generation of robot sequences, we randomly select stations with the robot that have been installed, and then randomly select a different type of robot to replace to obtain a new robot sequence $Sq_{robot}$.

In SA, as we know, local optimality can be escaped by accepting inferior solutions. As the temperature decreases, the probability of accepting the inferior solution becomes smaller, and the easier it is to fall into the local optimality. To address this issue, we designed

a restart mechanism with reference to Li et al. [36]. If the optima are not improved in $dn_{max}$ consecutive iterations, the algorithm returns to the initial solution generation phase.

### *4.5. Improvement Mechanism*

In this section, we propose a novel improvement rule, which embeds the exact algorithm into the SA algorithm to improve the quality of the solution.

As we know, the MILP model can be solved using exact algorithms or heuristic algorithms. Exact algorithms can find the optimal solution to the model, but for the complex assembly line problem, it is difficult to obtain a feasible solution when the allowable time is limited. Therefore, researchers generally use heuristic algorithms to obtain an approximate optimal solution. However, heuristic algorithms tend to fall into the local optimality, and the gap between the approximate optimal solution and the actual optimal solution cannot be measured.

In the improvement mechanism, we cut the original problem into a few small problems to find a better solution using exact algorithms. That is, most of the variables in RALBP are fixed, and only the remaining variables are relaxed. Based on the SA results, we select relaxed tasks and robots based on the three rules given in Table 3. In Example 3, the effectiveness of the improvement mechanism is shown.

**Table 3.** Improvement rules.

| | **Rules Description** |
|---|---|
| Rule 1 | The robots and tasks assigned to the last station are relaxed |
| Rule 2 | Compare the first 3 approximate optimal solutions of SA, the task and robot of assigning different positions are relaxed |
| Rule 3 | 10% of the tasks and robots are randomly relaxed in the remaining sequence (upper limit rounding) |

**Example 3.** *To test the effectiveness of the improvement mechanism, we compare the results before and after the introduction of the improvement mechanism. The sub-problems are derived from the datasets described in Section 5.1. The results of the comparison are shown in Figure 5. In Figure 5, the same color represents the same cycle time in datasets (Arcus, Heskiaoff, Scholl). Obviously, after the introduction of the improvement mechanism, although m has not changed, TEC has become smaller for datasets of different sizes. Thus, the improvement mechanism we propose is effective.*

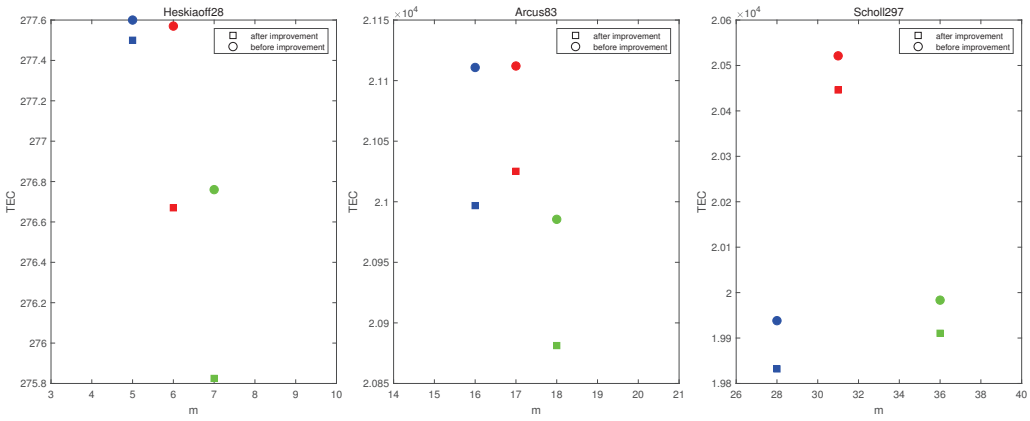


**Figure 5.** Example 3.

## 5. Computational Experiments

### *5.1. Design of Experiment*

The basic datasets are extracted from a well-known database (https://assembly-line-balancing.de/, accessed on 30 August 2022). They are Heskiaoff (28), Kilbrid (45), Arcus (83), and Scholl (297). The numbers inside the parentheses indicate the total number of

tasks of that dataset. The task time $t_{ir}$ is randomly generated based on the original data according to the fact that the higher the energy consumption, the lower the efficiency. $SPC_r$ and $OPC_r$ are selected by referring to Nilakantan et al. [27], which are provided in the supplementary file. For each instance, $c$ is fixed at six different values, and $\gamma$ is set to the $0.1 \times c$. Hence, there is a total of 24 independent experiments to conduct.

The particle swarm algorithm (PSO) and the late acceptance hill-climbing algorithm (LAHC) are two traditional methods that are compared. To observe the quality of each algorithm, we used the Gurobi 9.1.2 optimizer to solve the MILP model. Due to the excessive time of the exact algorithm for solving large instances, the runtime limit is set to 3600 s and the gap value is returned. To ensure that the heuristic algorithms are comparable, constrain the algorithm runtime ($rt$) to $rt = 10 \times n$ seconds. These algorithms are implemented in Matlab (R2019a) and executed on an AMD Ryzen 55500U 2.10 GHz CPU.

We have seen from the preliminary experiment results that the optima are not parameter sensitive. As a result, the parameter values are taken from the literature, as in Table 4. It should be noted that the initial temperature is case-dependent. The initial temperature in SA is determined using the methods described in Li et al. [36].

**Table 4.** Parameter values for each algorithm.

| Parameters | SA | LAHC | PSO |
|---|---|---|---|
| Cooling rate | 0.9 | - | - |
| Length of cost list | - | 100 | - |
| Learning coefficient $l_1(l_2)$ | - | - | 2(2) |
| The number of task(robot) particles | - | - | 30(30) |

### *5.2. Results and Analysis*

The computational results are displayed in Table 5. The best results each algorithm can obtain are recorded in column *Obj*. *Obj* is calculated by Formula (1), which contains information in both $m$ and $TEC$. Due to $TEC/TEC_{max}$ is in the range [0, 1], before and after the decimal point are our primary objective ($m$) and secondly objective ($TEC$), respectively. When the runtime is reached but the Gurobi optimizer cannot return a result, it is denoted by –. The unique optimal result between the three heuristics is marked in bold.

**Table 5.** Computational results.

| Dataset | c | Gurobi | | SA | PSO | LACH |
|---|---|---|---|---|---|---|
| | | Obj | Gap (%) | Obj | Obj | Obj |
| Heskiaoff | 160 | 7.7150 | 0.0 | **7.7153** | 7.7236 | 7.7235 |
| | 190 | 6.6989 | 0.0 | 6.7057 | **6.7041** | 6.7085 |
| | 220 | 5.6799 | 0.0 | 5.6848 | 5.6857 | **5.6832** |
| | 250 | 4.7058 | 0.0 | 5.6789 | **5.6782** | 5.6860 |
| | 280 | 4.6507 | 0.0 | **4.6518** | 4.6559 | 4.6548 |
| | 310 | 4.6435 | 0.0 | **4.6449** | 4.6538 | 4.6590 |
| Kilbrid | 70 | 8.7203 | 0.0 | **8.7528** | 8.7739 | 9.7163 |
| | 90 | 7.6807 | 0.0 | **7.6842** | 7.6889 | 7.6863 |
| | 110 | 5.6852 | 0.0 | **5.7273** | 5.7293 | 6.6619 |
| | 130 | 5.6313 | 0.0 | 5.6328 | **5.6334** | 5.6364 |
| | 150 | 4.6068 | 0.0 | **4.6105** | 4.6123 | 4.6190 |
| | 170 | 4.5893 | 0.0 | **4.5904** | 4.5906 | 4.5935 |
| Arcus | 4200 | 25.7511 | 52.6 | **18.7532** | 19.7510 | 19.7233 |
| | 4500 | – | – | 17.7538 | **17.7517** | 18.7151 |
| | 4800 | 17.7018 | 58.6 | **16.7229** | 16.7377 | 17.7113 |
| | 5100 | – | – | **15.7267** | 15.7277 | 16.7014 |
| | 5400 | – | – | **14.7218** | 14.7395 | 15.6946 |
| | 5700 | 14.6807 | 35.4 | **14.6866** | 14.7082 | 14.6911 |

**Table 5.** *Cont.*

| Dataset | c | Gurobi | | SA | PSO | LACH |
|---|---|---|---|---|---|---|
| | | Obj | Gap (%) | Obj | Obj | Obj |
| Scholl | 2000 | – | – | **36.7751** | 36.8018 | 37.7644 |
| | 2300 | – | – | 31.7916 | **31.7875** | 32.7511 |
| | 2600 | – | – | **28.7415** | 28.7516 | 28.7494 |
| | 2900 | – | – | **25.7373** | 25.7472 | 26.7213 |
| | 3200 | – | – | **22.7629** | 22.7809 | 23.7187 |
| | 3500 | – | – | **21.7004** | 21.7137 | 21.7110 |

Compared with three heuristic algorithms, the Gurobi optimizer returns the optimal solution for each instance (gap = 0) of the first two datasets. Thus, we can conclude that exact algorithms can quickly get a feasible solution and is optimal for instances with a small number of tasks. Additionally, 9 out of the 12 primary objective $m$ of the first two datasets are the same with three heuristic algorithms, indicating that the heuristic can find solutions as well as the exact algorithm for small instances.

However, when solving a problem with a large number of tasks, exact algorithms may find a solution with a large gap (Arcus) or not even find any feasible solution (Scholl), and the quality of the feasible solution may also be worse than that of the heuristic algorithms. Therefore, for large ALBP problems, heuristics are better than the exact algorithm.

Comparing the results among the three heuristic algorithms, the SA algorithm finds the best optimal solution where 18 out of 24 are the unique best, dominated by PSO in 5 instances, and is dominated by LACH in 1 instance. For the SA algorithm, the number of stations $m$ is optimal among the three heuristics. In addition, in the first dataset, the SA algorithm finds the best optimal solutions where 3 out of 6 are the unique best. The SA algorithm finds the best optimal solutions where 5 out of 6 are the unique best in the remaining three datasets. Thus, the SA algorithm is much better for solving large instances.

## 6. Conclusions

A satisfactory ecological environment is an important part of people's pursuit of a better life. Pollution from energy consumption in the industrial sector is a problem that cannot be ignored. Presently, robotic assembly lines are widely applied in industrial production. Though the introduction of multi-functional robots on assembly line results in a significant improvement in production efficiency, it brings about high energy consumption. Thus, how to balance energy consumption and efficiency is the goal of our paper.

In this paper, a robotic assembly line balancing problem considering minimizing the number of workstations as the primary objective and energy consumption as the secondary objective is investigated. Our research is the first attempt to model and solve the type-1 RALBP with multi-objectives and cross-station task design. A mixed integer linear integer programming model is formulated to solve the problem. A simulated annealing algorithm which encapsulates an improvement mechanism, is designed and compared with the particle swarm algorithm and the late acceptance hill climbing algorithm. The computational study shows that SA performs better than PSO and LAHC.

This study has some limitations. Since it is the first attempt to explore this innovative research topic, a simple single product is assumed, which is less useful than a general multi-product assumption. Additionally, we only represented three classic algorithms (SA, PSO, and LAHC) to solve the straight assembly line. Algorithmic design and varieties can be further improved to solve more complex assembly line balancing problems (two-sided, U-shaped, and parallel).

**Author Contributions:** Conceptualization, Y.L. and Y.C.; methodology, Z.Q.; investigation, M.L. and Y.Z. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was supported in part by the National Natural Science Foundation of China under Grant (71901006, 72140001), and the Humanities and Social Sciences of Ministry of Education Planning Fund of China (21YJA790009).

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The basic datasets at https://assembly-line-balancing.de/, accessed on 30 August 2022.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Li, Z.; Dey, N.; Ashour, A.S.; Tang, Q. Discrete cuckoo search algorithms for two-sided robotic assembly line balancing problem. *Neural Comput. Appl.* **2018**, *30*, 2685–2696. [CrossRef]
2. Fysikopoulos, A.; Anagnostakis, D.; Salonitis, K.; Chryssolouris, G. An empirical study of the energy consumption in automotive assembly. *Procedia CIRP* **2012**, *3*, 477–482. [CrossRef]
3. Bryton, B. *Balancing of a Continuous Production Line*; Northwestern University: Evanston, IL, USA, 1954.
4. Rubinovitz, J.; Bukchin, J. Design and balancing of robotic assembly lines. In Proceedings of the Fourth World Conference on Robotics Research, Pittsburgh, PA, USA, 17–19 September 1991.
5. Bock, S.; Boysen, N. Integrated real-time control of mixed-model assembly lines and their part feeding processes. *Comput. Oper. Res.* **2021**, *132*, 105344. [CrossRef]
6. Kilincci, O. Firing sequences backward algorithm for simple assembly line balancing problem of type 1. *Comput. Ind. Eng.* **2011**, *60*, 830–839. [CrossRef]
7. Manavizadeh, N.; Hosseini, N.S.; Rabbani, M.; Jolai, F. A Simulated Annealing algorithm for a mixed model assembly U-line balancing type-I problem considering human efficiency and Just-In-Time approach. *Comput. Ind. Eng.* **2013**, *64*, 669–685. [CrossRef]
8. Li, Z.; Tang, Q.; Zhang, L.P. Two-sided assembly line balancing problem of type I: Improvements, a simple algorithm and a comprehensive study. *Comput. Oper. Res.* **2017**, *79*, 78–93. [CrossRef]
9. Li, Y.; Wen, M.; Kang, R.; Yang, Z. Type-1 assembly line balancing considering uncertain task time. *J. Intell. Fuzzy Syst.* **2018**, *35*, 2619–2631. [CrossRef]
10. Li, Y.; Hu, X.; Tang, X.; Kucukkoc, I. Type-1 U-shaped Assembly Line Balancing under uncertain task time. *IFAC-Pap.* **2019**, *52*, 992–997. [CrossRef]
11. Zhang, H. An immune genetic algorithm for simple assembly line balancing problem of type 1. *Assem. Autom.* **2019**, *39*, 113–123. [CrossRef]
12. Baskar, A.; Xavior, M.A. Heuristics based on Slope Indices for Simple Type I Assembly Line Balancing Problems and Analyzing for a Few Performance Measures. *Mater. Today Proc.* **2020**, *22*, 3171–3180. [CrossRef]
13. Pınarbaşı, M.; Alakaş, H.M. Assembly line balancing type-1 problem with assignment restrictions: A constraint programming modeling approach. *Pamukkale Univ. J. Eng. Sci.* **2021**, *27*, 532–541. [CrossRef]
14. Huang, D.; Mao, Z.; Fang, K.; Yuan, B. Combinatorial Benders decomposition for mixed-model two-sided assembly line balancing problem. *Int. J. Prod. Res.* **2022**, *60*, 2598–2624. [CrossRef]
15. Rubinovitz, J.; Bukchin, J.; Lenz, E. RALB-A Heuristic Algorithm for Design and Balancing of Robotic Assembly Lines. *CIRP Ann.-Manuf. Technol.* **1993**, *42*, 497–500. [CrossRef]
16. Hong, D.S.; Cho, H.S. Generation of robotic assembly sequences with consideration of line balancing using simulated annealing. *Robotica* **1997**, *15*, 663–673. [CrossRef]
17. Gao, J.; Sun, L.; Wang, L.; Gen, M. An efficient approach for type II robotic assembly line balancing problems. *Comput. Ind. Eng.* **2009**, *56*, 1065–1080. [CrossRef]
18. Janardhanan, M.N.; Li, Z.; Bocewicz, G.; Banaszak, Z.; Nielsen, P. Metaheuristic algorithms for balancing robotic assembly lines with sequence-dependent robot setup times. *Appl. Math. Model.* **2019**, *65*, 256–270. [CrossRef]
19. Sun, B.Q.; Wang, L. An estimation of distribution algorithm with branch-and-bound based knowledge for robotic assembly line balancing. *Complex Intell. Syst.* **2021**, *7*, 1125–1138. [CrossRef]
20. Aslan, Ş. Mathematical model and a variable neighborhood search algorithm for mixed-model robotic two-sided assembly line balancing problems with sequence-dependent setup times. *Optim. Eng.* **2022**, 1–28. [CrossRef]
21. Michels, A.S.; Lopes, T.C.; Sikora, C.G.S.; Magatão, L. The Robotic Assembly Line Design (RALD) problem: Model and case studies with practical extensions. *Comput. Ind. Eng.* **2018**, *120*, 320–333. [CrossRef]
22. Pereira, J.; Ritt, M.; Vásquez, Ó.C. A memetic algorithm for the cost-oriented robotic assembly line balancing problem. *Comput. Oper. Res.* **2018**, *99*, 249–261. [CrossRef]
23. Rabbani, M.; Behbahan, S.Z.B.; Farrokhi-Asl, H. The Collaboration of Human-Robot in Mixed-Model Four-Sided Assembly Line Balancing Problem. *J. Intell. Robot. Syst. Theory Appl.* **2020**, *100*, 71–81. [CrossRef]
24. Koltai, T.; Dimény, I.; Gallina, V.; Gaal, A.; Sepe, C. An analysis of task assignment and cycle times when robots are added to human-operated assembly lines, using mathematical programming models. *Int. J. Prod. Econ.* **2021**, *242*. [CrossRef]

25. Lahrichi, Y.; Damand, D.; Deroussi, L.; Grangeon, N.; Norre, S. Investigating two variants of the sequence-dependent robotic assembly line balancing problem by means of a split-based approach. *Int. J. Prod. Res.* **2022**, 1–17. [CrossRef]
26. Nilakantan, J.M.; Huang, G.Q.; Ponnambalam, S.G. An investigation on minimizing cycle time and total energy consumption in robotic assembly line systems. *J. Clean. Prod.* **2015**, *90*, 311–325. [CrossRef]
27. Nilakantan, J.M.; Li, Z.; Tang, Q.; Nielsen, P. Multi-objective co-operative co-evolutionary algorithm for minimizing carbon footprint and maximizing line efficiency in robotic assembly line systems. *J. Clean. Prod.* **2017**, *156*, 124–136. [CrossRef]
28. Zhang, Z.; Tang, Q.; Zhang, L. Mathematical model and grey wolf optimization for low-carbon and low-noise U-shaped robotic assembly line balancing problem. *J. Clean. Prod.* **2019**, *215*, 744–756. [CrossRef]
29. Zhou, B.; Wu, Q. Decomposition-based bi-objective optimization for sustainable robotic assembly line balancing problems. *J. Manuf. Syst.* **2020**, *55*, 30–43. [CrossRef]
30. Zhang, B.; Xu, L.; Zhang, J. Balancing and sequencing problem of mixed-model U-shaped robotic assembly line: Mathematical model and dragonfly algorithm based approach. *Appl. Soft Comput.* **2021**, *98*, 106739. [CrossRef]
31. Belkharroubi, L.; Yahyaoui, K. Solving the energy-efficient Robotic Mixed-Model Assembly Line balancing problem using a Memory-Based Cuckoo Search Algorithm. *Eng. Appl. Artif. Intell.* **2022**, *114*, 105112. [CrossRef]
32. Grzechca, W.; Foulds, L.R. The assembly line balancing problem with task splitting: A case study. *IFAC-Pap.* **2015**, *28*, 2002–2008. [CrossRef]
33. Nanda, R.; Scher, J.M. Assembly lines with overlapping work stations. *AIIE Trans.* **1975**, *7*, 311–318. [CrossRef]
34. Kirkpatrick, S.; Gelatt, C.D.; Vecchi, M.P. Optimization by simulated annealing. *Science* **1983**, *220*, 671–680. [CrossRef]
35. Khorasanian, D.; Hejazi, S.R.; Moslehi, G. Two-sided assembly line balancing considering the relationships between tasks. *Comput. Ind. Eng.* **2013**, *66*, 1096–1105. [CrossRef]
36. Li, Y.; Kucukkoc, I.; Tang, X. Two-sided assembly line balancing that considers uncertain task time attributes and incompatible task sets. *Int. J. Prod. Res.* **2021**, *59*, 1736–1756. [CrossRef]