mathematics

MDPI

Article

A Hybrid Genetic Algorithm for the Simple Assembly Line Balancing Problem with a Fixed Number of Workstations

Eduardo Álvarez-Miranda [1], Jordi Pereira [2,*], Harold Torrez-Meruvia [3] and Mariona Vilà [3]

[1] School of Economics and Business, Universidad de Talca, Talca 3460000, Chile; ealvarez@utalca.cl
[2] Faculty of Engineering and Sciences, Universidad Adolfo Ibáñez, Av. Padre Hurtado 750, Viña del Mar 2520000, Chile
[3] EAE Business School, C. Aragó 55, 08015 Barcelona, Spain; harold.torrez@eae.es (H.T.-M.); mvila@eae.es (M.V.)
* Correspondence: jorge.pereira@uai.cl

Abstract: The assembly line balancing problem is a classical optimisation problem whose objective is to assign each production task to one of the stations on the assembly line so that the total efficiency of the line is maximized. This study proposes a novel hybrid method to solve the simple version of the problem in which the number of stations is fixed, a problem known as SALBP-2. The hybrid differs from previous approaches by encoding individuals of a genetic algorithm as instances of a modified problem that contains only a subset of the solutions to the original formulation. These individuals are decoded to feasible solutions of the original problem during fitness evaluation in which the resolution of the modified problem is conducted using a dynamic programming based approach that uses new bounds to reduce its state space. Computational experiments show the efficiency of the method as it is able to obtain several new best-known solutions for some of the benchmark instances used in the literature for comparison purposes.

Keywords: assembly lines; manufacturing; line balancing; hybrid genetic algorithm

Citation: Álvarez-Miranda, E.; Pereira, J.; Torrez-Meruvia, H.; Vilà, M. A Hybrid Genetic Algorithm for the Simple Assembly Line Balancing Problem with Fixed Number of Workstations. *Mathematics* **2021**, *9*, 2157. https://doi.org/10.3390/math9172157

Academic Editor: Frank Werner

Received: 6 August 2021
Accepted: 30 August 2021
Published: 4 September 2021

1. Introduction

The Assembly Line Balancing Problem (ALBP) is a classic problem that has been a subject of research for nearly seventy years [1]; see [2–4] for reviews on the problem. The objective of the problem is to assign each task into which the production process may be divided into one of the stations of the assembly line.

The most studied case is known as the Simple Assembly Line Balancing Problem (SALBP) [5]. This case considers serially arranged stations and a single product, which travels from station to station by means of a conveyor belt or a similar transportation device. The operations necessary to manufacture the product are known as tasks. Each task has a deterministic duration and may not be able to start until another task is completed. The latter condition is known as a precedence relationship, e.g., the seats of a car must be assembled before installing the doors. Each station performs one or several tasks, which generate a workload equal to the sum of the durations of all the tasks assigned to it. Once steady-state manufacturing conditions have been achieved, the production items travel along the line at a constant rate, and each station is allotted an identical time to complete its workload. This time is known as the cycle time, c, and it is equal to the maximum workload. The difference between the cycle time and the workload in each station is known as the idle time of the station. The objective of the problem is to assign the tasks to the stations in such a manner that maximises productivity, which is equivalent to minimising the sum of idle times for all the stations, while fulfilling workload constraints and precedence relationships.

Any other problem in which additional constraints or different objectives are considered is known in the literature as a General Assembly Line Balancing Problem (GALBP) [5].

While most real-life balancing problems belong in the GALBP family, the SALBP is the underlying basic formulation of many of these cases. In addition to the real-life importance, the SALBP provides a unified framework for algorithm comparison and it also acts as a test-bed to propose new ideas applicable to a wide range of balancing problems.

The case studied in the present work is the SALBP-2 in which, given a number of stations, maximising efficiency is achieved by minimising the cycle time. This case matches a line rebalancing problem which has been specifically studied in [6,7] among others, as it considers the number of stations from a previous line design, but technology or production changes alter the duration of the tasks, forcing the rebalancing of the line.

The literature is more focused on the study of the line design problem (SALBP-1), where the cycle time is known and the objective is to minimise the number of stations, even if line rebalancing problems (SALBP-2) are more frequent in real situations. The reason given by some authors [3,8] is that the SALBP-2 can be reformulated as the problem of finding the smallest cycle time for which the optimal solution to the corresponding SALBP-1 has the desired number of stations. This method has proved to be very effective, and resolution procedures using this property are collectively known as reformulation methods, as opposed to direct resolution methods.

As stated in [3],

> The success of reformulation methods "gives a certain foundation for the concentration on finding effective procedures for SALBP-1 in the past 50 years though SALBP-2 is a problem which may even have the greater practical relevance because it arises whenever an existing line has to be rebalanced, while SALBP-1 is relevant mainly in case of the first installation of a line."

The previous statement is supported by the state-of-the-art methods found in the literature, which are based on reformulation approaches, but hard SALBP-2 instances still seem to be more challenging than their SALBP-1 counterparts. While the SALBP-1 reference set contains no open instances, the reference set for SALBP-2 contains 14 open instances out of 302.

This perceived difficulty could be explained by the fact that a reformulation method is required to solve several SALBP-1 instances, including the instance with the smallest possible total idle time and the desired number of stations. Such an instance is more difficult than any other instance with greater idle time and the same number of stations, as the set of solutions of the former is a subset of the set of the latter. Moreover, an exact reformulation method also needs to verify that there is no feasible assignment for the instance with a cycle time that is one unit lower.

Both the practical relevance and the greater perceived difficulty of the SALBP-2 drive us to the study of this problem and the development of a new algorithm to solve hard instances. The proposed method is based on a SALBP-1 solution procedure but modifies the search space to add additional exploration in order to tackle these more challenging instances.

1.1. Problem Description

Formally, a SALBP-2 instance is defined by a set of elementary tasks V, with $|V| = n$, into which the assembly process of the product has been divided and a set of m workstations. Each task has a known deterministic duration, d_i, $i = 1, \ldots, n$, and it must be assigned to one of the stations S_j, $j = 1, \ldots, m$. Precedence relations between tasks are represented using an acyclic graph $G(V, A)$ in which the vertices represent each one of the tasks and an arc between two vertices denotes a precedence relationship between the tasks. The task associated with the initial vertex is commonly referred to as the predecessor task and must be processed before the task associated with the terminal vertex, commonly referred to as the successor task.

The objective of the problem is to assign the tasks to the stations minimising the cycle time, c, defined as the maximum workload of any station, while fulfilling all the precedence constraints.

Given a set of binary decision variables x_{ij} associated with assigning task i to station j, objective (1) and constraint sets (2)–(4) represent a valid integer programming model for the SALBP-2:

$$[\text{MIN}]c = \max_{1 \leq j \leq m} \sum_{i=1}^{n} d_i x_{ij}, \tag{1}$$

$$\sum_{j=1}^{m} x_{ij} = 1 \qquad 1 \leq i \leq n, \tag{2}$$

$$\sum_{j=1}^{m} j x_{ij} \leq \sum_{j=1}^{m} j x_{i'j} \qquad (i, i') \in A, \tag{3}$$

$$x_{ij} \in \{0, 1\} \qquad 1 \leq i \leq n, 1 \leq j \leq m. \tag{4}$$

The objective function (1) minimises the cycle time. Constraint set (2) ensures the unique assignment of each task to a station, constraint set (3) ensures the fulfilment of precedence constraints, and (4) defines the decision variables as binary. Note that this problem is reversible. As such, the direction of the precedence constraints may be reversed, and a solution for the original instance can be obtained by renumbering the stations of the solution to the reverse instance, $S_j \leftarrow S_{m+1-j}$, for every S_j, $j = 1, \ldots, m$.

1.2. Review on the Resolution Methods

As previously discussed, the methods to solve the SALBP-2 can be classified as either reformulation or direct methods. A secondary classification divides them into heuristic and exact methods. A tertiary classification applicable for reformulation methods divides them according to how they explore the range of cycle times. As numerous procedures have been proposed in the literature (see [3] for a review of solution methods), we only cite some of the most significant approaches for the resolution of the SALBP-2 and GALBP with cycle time minimization objectives and refer to the references in [2,3] for additional details.

Among the direct approaches, we highlight the Tabu Search (TS) heuristic proposed in [9], the two-phase heuristic based on linear programming proposed in [10], the variable neighborhood search, simulated annealing hybrid proposed in [11], the iterated local search proposed found in [12] the exact method described in [13] and the constraint programming approaches described in [14,15].

Heuristic approaches initially find a feasible solution using a greedy heuristic for the SALBP-1 and then explore the neighbourhood of the SALBP-2 solution found during the initial step. The TS [9] explores the neighbourhood using two local search operators: the first moves a task from one station to another, and the second exchanges the station assignment of two tasks. The linear programming based method [10] explores a different neighbourhood using a variant of the simplex algorithm to search through the vertices of a polytope containing integral solutions to the problem's integer programming formulation. The methods proposed in [11,12] take into account other features from the GALBP problem under study and define different neighborhoods that not only take tasks into account but also other features of the problem, like sides within the assembly line, or stochastic task times.

The exact method described in [13] proposes a Local Lower Bound Method enumeration scheme for the problem. In comparison to the reformulation methods, its main advantage is that it maintains a single enumeration tree during the complete search. Its main drawback is that better lower bounds exist for the SALBP-1.

Among the heuristic methods based on a reformulation approach, we highlight a Petri-net-based heuristic [16], an evolutionary algorithm based on the Differential Evolution (DE) paradigm [17], and a constructive iterative Beam Search (BS) heuristic [18]. All three methods use trial cycle times and iteratively solve the related SALBP-1, but they differ on the technique used to tackle the SALBP-1 and the order in which the trial cycle times are explored.

The heuristic proposed in [16] uses an initial trial time equal to a lower bound on the optimal cycle time, and it increases the cycle time until a feasible solution is found. This technique for the exploration of the interval of cycle times is known as a lower bound method [13]. Each SALBP-1 instance is solved using a constructive procedure based on a Petri Net token movement and reachability analysis. Ref. [19] uses a similar approach but codes the solution as a permutation of tasks that need to be decoded to obtain an assignment of tasks to stations.

The DE algorithm proposed in [17] codes each solution using a random-key for each task. Each DE solution is evaluated using a station-oriented SALBP-1 greedy heuristic that uses the random-key as the priority value for each task. The interval of cycle times is also explored using the lower bound method. The indirect solution representation is also used in [20–22].

The iterative BS [18] uses the Beam Search heuristic to solve the trial values. The search is divided into two phases. The first phase uses a lower bound method to find a feasible cycle time. This phase is set to use a small fraction of the total allotted time. The second phase uses an upper bound method, in which the algorithm tries to find a valid solution for the next possible smaller cycle time until the total run time is consumed.

Lastly, Ref. [8] provides results of the iterative application of SALOME-1 (an exact solution method for the SALBP-1) to optimally solve the problem. The cycle time search interval is successively divided into subintervals using a binary search until the lower bound is equal to the upper bound.

The cycle time minimization objective of the SALBP-2 is a common objective within multi-objective versions of the SALBP [23] and the GALBP [21,24,25], assembly line balancing problems with heterogeneous workstations [19,26–29].

1.3. Outline of the Proposed Algorithm

This study proposes a new hybrid method intended to address hard SALBP-2 instances. The hybrid procedure is a reformulation method and combines a genetic algorithm (GA) [30] with an adapted and strengthened version of one of the current state-of-the-art heuristic procedures for the SALPB-1, the Dynamic Programming (DP)-based Bounded Dynamic Programming (BDP) procedure [31].

The BDP is a resolution procedure based on the partial exploration of the state-space of the problem. Given the very large number of states of any DP formulation for the problem, it is impractical to perform a complete enumeration of the state-space associated with the DP. This is why the BDP explores a reduced state-space by (1) using lower bounds and dominance rules to purge states and (2) using heuristic rules to discard the least promising states.

The presented algorithm tries to alleviate one of the shortcomings of the BDP when solving difficult SALBP-1 instances, namely, the use of a heuristic rule to discard states that favour intensification over diversification. This leads to disregarding large areas of the state-space that may contain the optimal solution for difficult instances, as shown in the computational experiments.

In this work, diversification is introduced into the search by incorporating new constraints into the original problem. The addition of constraints modifies the state-space, leading the BDP to explore a subset of the original search space.

The constraints incorporated into the original problem are incompatibilities between tasks. An incompatibility between two tasks indicates that these tasks cannot be performed in the same station. This problem has been studied previously in the literature (see the review of previous work in [32], and it is also studied in the present work to develop new lower bounds and reduction rules. Note that, as a consequence, the present manuscript makes several contributions to the state-of-the-art of the problem with incompatibilities between tasks, even if it is not the main focus of this work. From this point forward, this problem will be denoted as the IBT-ALBP (Incompatibilities Between Tasks-Assembly Line Balancing Problem).

The generation of these modified instances is performed by the GA component of the procedure. A GA is a metaheuristic method inspired by natural selection and the evolution of species, commonly used for solving difficult optimisation problems. A regular GA explores the solution space of the problem using a set of solutions, known as the population of individuals. The fitness of each individual is related to the quality of the solution it represents, and it is used to decide which individuals create a new population using different operators (i.e., selection, crossover, and mutation).

In our case, each individual of the GA represents an IBT-ALBP instance. The GA is hybridised with the BDP method, which decodes individuals into SALBP-2 solutions while measuring their fitness.

The results obtained in the computational experiments carried out with the algorithm show the advantages of the proposed method. The presented algorithm improves the best-known solution for eight of the fourteen open instances from the benchmark instance set and obtains the best-known solution for all but one of the instances in the reference set.

The remainder of the study is structured as follows. Section 2 introduces the IBT-ALBP, presents some new lower bounds and pre-processing techniques for the problem, and puts forward the modified BDP method proposed for its resolution. Section 3 presents the hybrid GA procedure. Section 4 discusses the results of a computational experiment conducted with the algorithm. Lastly, Section 5 puts forward the conclusions derived from the present work.

2. Line Balancing with Incompatibilities between Tasks

In this section, we study the IBT-ALBP. A description of the problem is given in Section 2.1, whereas subsection 2.2 studies some lower bounds for the problem. Lastly, the BDP method proposed to solve the IBT-ALBP is described in Section 2.3.

2.1. Problem Description

As stated in Section 1, the objective of the SALBP-2 is to assign production tasks to stations while fulfilling certain precedence constraints and maximising the efficiency of the line. Other constraints may be added to this basic formulation to obtain more general problems. One of these additional constraints often found in real-life assembly line balancing problems are assignment restrictions, see [32] for a classification—which model zoning constraints or resource restrictions, among others. Incompatibilities between tasks are among the most common representations of these restrictions.

An incompatibility constraint between two tasks forces them to be processed by two different stations. The problem has been previously studied by different authors (see [32–35] among others) including several real-life case studies [36,37].

The formulation of the problem builds upon the original SALBP formulation seen in Section 1, see (1)–(4), where constraint set (5) is added. An auxiliary graph $G'(V,E)$ is used to represent the incompatibilities. The vertices of the graph represent the tasks, and an edge between two vertices indicates that the associated tasks cannot be processed in the same station,

$$x_{ij} + x_{i'j} \leq 1 \qquad \{i,i'\} \in E,\ j = 1,\dots,m. \tag{5}$$

There are three common characteristics between the original and the modified problem that are used in the present work:

1. The problem may feature both type-1 and type-2 objectives, and the latter can be solved using a reformulation method;
2. The SALBP is a simplified version of the IBT-ALBP and, as such, any lower bound for the SALBP is a valid lower bound for the modified problem;
3. The problem is reversible (see Section 1.1).

2.2. Bounds and Reduction Rules

The study of lower bounds and reduction rules in this work is limited to the problem with a type-1 objective, as the reformulation method solves the type-2 problem by iteratively solving type-1 instances. These bounds and reduction rules will be used both before attempting to solve the problem to preprocess the instance and within the BDP to reduce the state-space.

2.2.1. SALBP-1 Lower Bounds

A first set of bounds is derived from the relationship between the SALBP and the IBT-ALBP. The procedure four classical lower bounds, which are referred to as LB_1, LB_2, LB_3, and LB_4 according to [8] notation.

The first three (LB_1, LB_2 and LB_3) of these bounds are an assimilation of a type-1 bin-packing problem. LB_1, also known as the trivial bound, corresponds to the sum of the durations of all of the tasks divided by the value of the cycle time. LB_2 considers the minimum number of workstations required by those tasks that require more than half the workstation workload, i.e., tasks where $d_i > c/2$ holds, and the tasks that require half the workstation workload, i.e., tasks with a duration of exactly half the cycle time. LB_3 behaves like LB_2 but considers tasks according to thirds of the cycle time.

The fourth bound (LB_4) relaxes the problem to a one-machine scheduling problem [38]. Tasks are interpreted as jobs, with operation times $p_i = \frac{d_i}{c}$ for all $i = 1, \ldots, n$ that must be performed on one machine. After processing each job, a certain amount of time is needed before the operation is finished. This time is known as the *tail* of the task. The objective of the problem is to find a sequence of jobs that minimises the makespan, whose value is a lower bound on the required number of stations.

The optimal solution of the one-machine problem can be obtained by processing the jobs in order of non-increasing tails. Let η_i, $i = 1, \ldots, n$ be the tails of the tasks and let $\langle h_1, \ldots, h_n]$ be such an ordering of tasks. Then, the minimum makespan can be obtained as in (6):

$$\eta_i = \max\{p_{h_1} + \eta_{h_1}, p_{h_1} + p_{h_2} + \eta_{h_2}, \ldots, p_{h_1} + p_{h_2} + \ldots + p_{h_n} + \eta_{h_n}\}. \tag{6}$$

Initially, not all of the tails are available, i.e., the available tails are only those that pertain to tasks with no successors and value 0. To compute the tail of the remaining tasks, one uses (6) considering only the set of successors of the said task. During the computation, the tail of task i, η_i, can be rounded up using the following two rules:

Rule 1: if both $\eta_i < \lceil \eta_i \rceil$ and $p_i + \eta_i > \lceil \eta_i \rceil$ hold, the argument for the rounding is that the task cannot share a station with its successors. This rule was originally proposed in Johnson (1988).

Rule 2: if $(i, i') \in G(V, A')$ $(i, i') \in G(V, E)$ and $\lfloor \eta_i \rfloor = \lfloor \eta_{i'} \rfloor$, then η_i can be rounded up. The rationale is as follows: if $\lfloor \eta_i \rfloor = \lfloor \eta_{i'} \rfloor$, the tails state that tasks i and i' may share a station. As the incompatibility forbids the assignment, and task i should be assigned before task i', and its tail can be rounded up following the argument described in rule 1. Note that this rounding rule is a novel contribution of this study.

To obtain a lower bound for the problem, the calculation determines the tail of a fictitious task that is a predecessor of all other tasks. Moreover, one can perform the same calculations on the reverse instance to obtain an alternative bound. The tails for the reverse instance are denoted as the heads (a_i) of the tasks, and refer to a certain amount of time that is needed before the operation is started.

The heads and the tails of each task also provide a bound on the earliest e_i and latest l_i station to which each task may be assigned if the solution is to have m stations (see Equations (7) and (8)). These values are useful for reducing the number of states of the dynamic program and for detecting implicit incompatibilities. Note that m is known in advance, as the number of desired stations m is part of the SALBP-2 instance definition:

$$e_i = \lfloor a_i \rfloor + 1, \tag{7}$$

$$l_i = m - \lfloor \eta_i \rfloor. \tag{8}$$

2.2.2. New Lower Bounds for the IBT-ALBP

A new bound can be derived exclusively from the incompatibilities. The bound is noted as LB_{IBT}, and it is based on the identification of strongly connected sub-graphs in $G(V, E')$ and the observation that each strongly connected sub-graph contains a set $V' \subseteq V$ of tasks that cannot share a station with any other task in the subset.

Theorem 1. *The cardinality of any strongly connected sub-graph in $G'(V, E)$ is a lower bound on the optimal solution of the type-1 IBT-ALBP.*

Proof of Theorem 1. In a strongly connected graph, every vertex is connected with every other vertex. As such, none of the associated tasks can share a station. Therefore, the minimum number of stations to which these tasks can be assigned is the cardinality of the subset of vertices that defines the strongly connected graph. □

Theorem 2. *Let $V' \subseteq V$ be a subset of tasks from $G'(V, E)$ that defines a strongly connected sub-graph and e_i be the earliest station to which task i may be assigned; then, $\min_{i \in V'}\{e_i + |V'| - 1\}$ is a valid lower bound on the number of stations for the type-1 IBT-ALBP.*

Proof of Theorem 2. No task in V' can be assigned to any station before $\min_{i \in V'}\{e_i\}$, and a minimum of $|V'| - 1$ additional stations are required to assign the remaining tasks in V'. □

Theorem 3. *Let $V' \subseteq V$ be a subset of tasks from $G'(V, E)$ that defines a strongly connected sub-graph. Let the tasks in V' be sorted by the non-decreasing earliest station. Assume that $\langle v_1, \ldots, v_{|V'|}$ represents such an ordering. Then, $\max_{i \in V'}\{e_{v_i} + |V'| - i\}$ is a valid lower bound on the optimum value of the problem.*

Proof of Theorem 3. For v_1, the bound provides the same result as Theorem 2. For v_2, the best possible circumstance is that the task associated with v_1 has been scheduled in a previous station, and thus the remaining problem would provide a bound using the proof for Theorem 2. The same reasoning holds for the remaining vertices. □

Based on these theorems, LB_{IBT} is defined as the maximum value reported by Theorem 3 for every strongly connected sub-graph in $G'(V, E)$ which are identified using [39].

2.2.3. Preprocessing Rules

Both the previous bound and the new rounding proposed for LB_4 depend on the number of incompatibilities between the tasks. While the original SALBP formulation does not contain explicit incompatibilities, instances may contain them implicitly, and their explicit representation can tighten the value provided by these bounds. Two methods are proposed:

The first method makes use of the earliest and latest station for each task. Let i and i' be two tasks with durations d_i and $d_{i'}$ and let the cycle time be c. If $d_i + d_{i'} > c$ holds, then tasks i and i' must be assigned to different stations, and the incompatibility may be added.

Note that, with the introduction of these incompatibilities, LB_{IBT} dominates LB_2 because it will provide an equal or better bound.

These two methods make use, directly or indirectly, of the durations, and thus a rule to increase them would improve their quality. In this case, we propose a strengthened version of the EDAR rule for the SALBP-1 [38].

The set of tasks that may share a station with any task i is composed by those tasks i' for which $e_{i'} \leq l_i$, $e_i \leq l_{i'}$, and no incompatibility between these tasks exists. If no combination between i and the tasks in the set fills a station completely, the duration of i, d_i, can be modified to account for the unavoidable idle time, i.e., the difference between the cycle time and the maximum load including task i. This formulation corresponds to a binary knapsack, where weights and values of each item correspond to the duration of the tasks (see [40]).

Note that any change in the durations of the tasks may also change the computation of LB_4 which in turn might change the earliest and latest station to which a task may be assigned. Each change may detect new incompatibilities, leading us to compute the aforementioned rules iteratively until no further changes appear.

2.3. Resolution by Means of Bounded Dynamic Programming

Dynamic programming, DP, approaches were one of the first methods used to solve ALBPs [41], and they were recently shown to be competitive for the SALBP-1 [31] and certain general cases [42] when the formulation is heuristically relaxed.

The DP formulation considers the SALBP-1 as a multistage decision process in which, for every stage u, the assignment of tasks to station u must be decided. Each stage contains a set of states that define all the possible partial solutions up to the stage. A state S is subsequently represented by a subset $S \subseteq V$ of assigned tasks. The states of stage $u+1$ can be generated from the states of stage u using a transition. A transition indicates which tasks are assigned to station $u+1$. A transition between two states S_u and S_{u+1} exists only if (9) and (10) hold:

$$\sum_{i \in S_{u+1} \setminus S_u} d_i \leq c, \tag{9}$$

$$i \notin S_{u+1} \implies i' \notin S_{u+1} \qquad \forall (i, i') \in A. \tag{10}$$

Any sequence of transitions taken from the initial state $S = \emptyset$ to the final state $S = V$ is a solution for the problem. The optimal solution is defined by the shortest sequence of transitions between these two states.

To use the DP formulation to solve the IBT-ALBP, it is necessary to modify the transitions. Transitions must verify that incompatible tasks are not assigned in the same station. Thus, for a transition to exist between S_u and S_{u+1} in addition to conditions (9) and (10), (11) and (12) must hold:

$$\text{if } i \notin S_u \wedge i \in S_{u+1} \implies i' \notin S_{u+1} \qquad (i, i') \in E, \tag{11}$$

$$\text{if } i \notin S_u \wedge i' \notin S_u \wedge i' \in S_{u+1} \implies i \notin S_{u+1} \qquad (i, i') \in E. \tag{12}$$

The major drawback of the previous DP formulation is that the number of transitions and states grows exponentially with the number of tasks. As such, a direct DP approach is often impractical. To handle these problems, the BDP applies several lower bounds, reduction techniques, and heuristic rules to reduce the number of states and transitions that need to be considered in each stage.

The BDP explores the state-space stage by stage, beginning with the initial stage, composed of a single state $S = \emptyset$. The method uses two lists into which the states are sequentially stored: a first list for the states of the last stage explored and a second list for the states of the stage under construction. Both lists are allowed to contain a single copy of each state.

Each state from the first list is removed, and the descending states, those that can be created from it after a transition, are stored in the second list. Only a subset of the descending states is generated to reduce the run time of the algorithm. Moreover, after generating all descending states, the list is subjected to a reduction procedure to remove the least promising states and then replaces the first list. The process is repeated until a solution is found or both lists are empty, in which case the algorithm reports the stage in which the last feasible state was constructed. Algorithm 1 outlines the BDP procedure.

Algorithm 1 requires the instance definition, the cycle time, c, and the desired number of the stations, m, as inputs and two parameters, w and t that are used to control the behaviour of the enumerate and reduce procedures. The first parameter w is the maximum number of states that the algorithm is allowed to maintain from one stage to the following. The second parameter t is the maximum number of transitions to develop for each state. These procedures are briefly described below.

Algorithm 1 Outline of the BDP procedure.

```
Input: Parameters $w$, $t$, $c$, and $m$, and instance definition
1:  Initialisation, $stage \leftarrow 0$, $States[stage] \leftarrow \{\emptyset\}$, $endCondition \leftarrow False$
2:  repeat
3:      $stage \leftarrow stage + 1$
4:      for $state \in States[stage - 1]$ do
5:          $States[stage] \leftarrow States[stage] \cup$ enumerate$(state, c, t)$
6:          if $\exists state \in States[Statage] \mid state = V$ then
7:              $endCondition \leftarrow True$                    ▷ Solution found
8:      $States[stage] \leftarrow$ reduce$(States[stage], w, m)$
9:      if $States[stage] = \{\emptyset\}$ then
10:         $endCondition \leftarrow True$                        ▷ no solution found
11: until $endCondition = True$
```

The enumerate procedure takes a state and enumerates every feasible task assignment for the station stage. The procedure returns the t "best" assignments enumerated (those with minimum idle time), breaking any possible ties in favour of those generated first. The above rule makes it possible to stop the enumeration when t assignments with zero idle time have been generated.

The enumerate procedure is implemented using an enumeration method reminiscent of the Hoffmann heuristic [43], which is adapted for the IBT-ALBP. To further reduce the enumeration effort, tasks whose latest feasible station, l_i, corresponds to the stage under construction are pre-assigned. This rule was not present in the original proposal to solve the SALBP-1 [31], and it reduces the computing time required by the method. In addition, note that enumerating will return a void set of assignments if the total duration of unassigned tasks with $l_i = stage$ is greater than the cycle time or $l_i = stage$ holds for two unassigned incompatible tasks.

The efficiency of the enumerate procedure depends on the order of the tasks. To increase the overall efficiency, the tasks are reordered so that (1) successors always appear after their predecessors, (2) tasks have a non-decreasing earliest station (e_i) order, and (3) if there is a tie in the previous rules, the task with a greater processing time appears first on the list. This ordering was previously proposed by several authors [8,31]. Note that different orderings might provide different solutions (due to the tie-breaking rule among states with the same idle time). This property is used in one of the hybrid methods proposed in the computational experience for comparison purposes.

The reduce procedure uses bounds LB_1, LB_3, and LB_{IBT} to prune the list by eliminating the states that report a lower bound on the required stations greater than m. Next, the procedure orders the remaining states in non-decreasing total idle time order and selects the first w different states or all states if the number of states is less than w. Non-selected states are discarded. Note that LB_4 and the preprocessing rules are only applied during the initialisation of Algorithm 1.

3. The Genetic Algorithm

A Genetic Algorithm (GA) [44] is a metaheuristic commonly used for solving difficult optimisation problems. The method is inspired by natural selection and the evolution of species. GAs are some of the most commonly used metaheuristics for solving ALBPs (see [45] for a review). According to [3], the adaptation of a GA to a specific ALBP needs to address two main issues: (1) the encoding of individuals and (2) the definition of an effective fitness function.

To tackle the first issue, previous methods have used one of the two following approaches:

- A direct encoding, in which the individuals are represented as vectors containing the labels of the station to which each task is assigned [46]. Alternatively, individuals can be represented as groups of tasks for each station [47,48].
- An indirect encoding, in which a secondary method, usually a task-oriented or a station-oriented constructive scheme, decodes the individual to obtain a solution. Several indirect encoding mechanisms have been proposed, such as using a sequence of tasks [49] or a priority value for each task [50].

Both direct and indirect encodings have their advantages and disadvantages. A direct encoding offers a direct exploration of the solution space, but standard crossover and mutation operators generate infeasible solutions. The GA will then need to address these infeasibilities directly, using a recovery operator, or indirectly, by penalising them during fitness evaluation. In contrast, indirect methods can use problem-specific algorithms to decode individuals, but the mapping between individuals and solutions is not unique, and, as such, several different encodings may lead to the same solution. Overall, indirect methods appear to offer better solutions for ALBPs (see [51,52] for indirect methods applied to some GALBPs).

The second issue, namely, the definition of a fitness function, revolves around the relationship between the objective and fitness functions. A GA needs a good fitness function to lead the exploration to promising areas of the solution space, but the objective functions of some ALBPs, particularly those with type-1 objectives, are not able to provide such an indication. This has led to the use of indirect fitness functions, such as using the imbalance of station loads [49,53].

For type-2 problems, the objective can be directly linked to fitness. While this direct relation is clearly advantageous, it may come at the price of wasted computation time. Most reformulation approaches [51,52], evaluate each individual using a greedy-like heuristic in combination with a lower- or upper-bound reformulation method, leading fitness evaluation to solve for trial cycle times that cannot improve the best-known solution. This may not be an issue when the evaluation procedure is a greedy-like heuristic, but, in our proposal, it needs to be addressed because each trial is solved using a BDP, which is a computing-intensive task. As such, an indirect objective function is more desirable than a direct one.

The proposed hybrid GA offers a novel approach to these two issues and uses simple methods to initialise, cross, and mutate the individuals. The method uses an indirect encoding method based on representing individuals as instances of the IBT-ALBP. Individuals are decoded using the BDP, but—to avoid the drawbacks of a direct fitness evaluation—a secondary function is used. This function measures the relative difficulty of the BDP in finding an improved solution.

The rest of the section is divided into four subsections. Section 3.1 is devoted to the encoding and the description of the procedure used to initialise individuals. Section 3.2 studies fitness evaluation and describes the reformulation method in which the hybrid GA is integrated. Section 3.3 presents the genetic operators not described in any previous subsection and introduces the parallelisation approach. Lastly, Section 3.4 gives an outline of the proposed method.

3.1. Representation Scheme and Initialisation

Indirect encoding methods explore a different solution space than direct methods. While both try to obtain good solutions to the problem, indirect methods search the best input data for a secondary search method that is then responsible for offering a solution to the problem. In our case, the secondary search method is the BDP proposed in Section 2.3, and each individual represents an IBT-ALBP instance. An individual will have a good fitness value (see Section 3.2) if the modified solution space created by the corresponding IBT-ALBP instance leads the BDP away from previously explored local optima and towards improved solutions.

Each individual is represented using a list of pairs of incompatible tasks that define the edges of graph $G'(V, E)$. To use the reversibility property, each individual also contains a binary value that defines the direction of the arcs of the precedence graph $G(V, A)$.

The list has a fixed length, equal to l, which is a parameter of the algorithm. Each pair contains two integers to identify the incompatible tasks. The binary value associated with the direction is equal to 0 if the original precedence graph should be used and 1 if the reverse is preferred.

Note that some of these incompatibilities may be irrelevant (e.g., a pair of tasks may already have an implicit incompatibility), and, as such, l should be seen as a maximum rather than the actual number of pairs added to the original instance. This effect is deliberately accepted to allow individuals with a different number of incompatibility pairs to exist within a single population, while keeping an upper limit to avoid an excessive perturbation of the original solution space.

Individuals of the initial population are randomly constructed as follows: for each incompatibility pair from 1 to l, randomly select (with even probability) two different tasks to form the pair. Note that repeated pairs are accepted. The value associated with the direction is randomly set to 0 or 1 with even probability.

3.2. Reformulation Method and Fitness Evaluation

A commonly used technique to embed the reformulation method within an evolutionary algorithm is to use it within fitness evaluation [17,51,52,54]. In such a case, the evaluation of an individual requires the exploration of the range of feasible cycle times, by means of a lower bound, upper bound, or binary search method, among others [8], using the decoding scheme as the evaluation method of each trial cycle time.

The main drawback of this technique is that it requires the resolution of several trial type-1 instances in each evaluation, some of which are only intended to provide a fitness value.

In contrast, in our proposal, the GA is embedded into the reformulation method. The new reformulation is divided into two phases. The first phase uses a binary search method to obtain an initial cycle time, whereas the second phase uses the GA in conjunction with the BDP to find improved solutions.

The first phase initially obtains a lower bound, $\underline{c}$, and an upper bound, $\overline{c}$, on the cycle time using (13) and (14), respectively

$$\underline{c} = \frac{\sum_{i \in V} d_i}{m}, \tag{13}$$

$$\overline{c} = \sum_{i \in V} d_i. \tag{14}$$

The interval $[\underline{c}, \overline{c}]$ of candidate cycle times is then explored using a binary search method in conjunction with the BDP proposed in Section 2.3. Note that (1) the original instance, with no incompatibilities between tasks, is solved during this phase; and (2) the BDP attempts to solve both the direct and reverse instance before marking a trial cycle time as unfeasible.

In the second phase, the hybrid GA tries to improve the best-known solution using an upper bound method. During fitness evaluation, the BDP tries to solve the IBT-ALBP instance coded in the individual using a cycle time one unit smaller than the best-known cycle time. If the BDP finds a feasible solution, the current best-known cycle time is updated, and the BDP is executed with a new cycle time. Fitness evaluation stops when the BDP is not able to find a feasible solution. At this point, the fitness of the individual is set to the last station for which the BDP was able to generate a partial assignment. The value is an indication of the difficulty the BDP encounters in finding an improving solution.

The main advantage of the approach is that it reduces the run time devoted to fitness evaluation, which is the most time-consuming element of the algorithm, at the price of losing the direct relationship between the objective function and fitness value.

Note that other similar indirect fitness functions are possible. Some of them were tested, such as counting the maximum number of tasks assigned to any partial solution or the total duration of tasks assigned to any partial solution, but they were deemed to be inferior in terms of solution quality to the fitness function used in the final implementation.

In addition, note that the fitness of different individuals may come from the evaluation of different trial cycle times (take the extreme case of an individual improving the best-known cycle time; once inserted into the population, it is the only individual evaluated with the new sought-after cycle time). While this behaviour appears to be counter-productive, our preliminary tests showed that this was not the case and that any method oriented to favour individuals who improved the best-known solution provoked an undesired level of elitism.

3.3. Genetic Operations and Parallelisation Scheme

In addition to the representation scheme and the initialisation and evaluation method of the individuals, an implementation of a GA needs to define a selection, crossover, mutation, and replacement operator and a stopping criterion. This subsection addresses the description of each of these elements and the parallelisation technique proposed to make use of the multi-core architecture of current commodity computers.

The selection operator is based on a binary tournament. To select each parent, two candidates from the current population are randomly chosen, and the better fit becomes the parent.

Two parents always produce a child using the crossover operator (i.e., crossover probability equals 1). The crossover randomly chooses characteristics from the parents. In this case, for each pair in the list of incompatibilities, from 1 to l, the operator selects (with even probability) the corresponding incompatibility pair from one of the parents. The direction of the precedence constraint is also evenly chosen between the directions coded in the parents.

The mutation operator uses a fixed mutation rate to diversify the search. For each incompatibility pair, the pair mutates with probability mp, where mp is a parameter of the algorithm. In the case of mutation, a new pair is generated by randomly selecting (with even probability) two different tasks to form a new pair that is substituted for the previous one. The direction is also subject to mutation with probability mp, and, if it mutates, the direction is reversed.

The child obtained after crossover and mutation is evaluated and is inserted in the population by replacing a randomly chosen member. The replacement operator also uses a binary tournament. Two candidates from the current population are chosen with even probabilities, and the worst fit candidate (lowest fitness value) is replaced.

The described GA corresponds to a steady-state GA in which each iteration only alters a subset of the total population. This scheme leads to an easy parallelisation using a master–slave model [55]. The implementation creates a thread for each available processor in the computer, and each thread works concurrently on the same population. Once the population has been initialised, each thread applies the selection, crossover, mutation, evaluation, and replacement operators concurrently until the stopping criterion is met. The stopping criterion used in the final implementation is a run time (CPU time) limit.

The proposed parallelisation method can be described as a coarse-grained parallelisation format. This scheme was chosen for two main reasons: (1) the most intensive task (fitness evaluation) is fully parallelised; and (2) synchronisation between different threads is only necessary during the selection and replacement operations—to avoid concurrent reading/writing operations—or to update the best-known solution.

Note that the implemented operators and the parallelisation scheme are simple when compared with some previous proposals [51,52]. This has been a deliberate decision, to highlight the importance of the two main contributions of this work: the use of a derived problem to code the individuals, and the hybridisation of the GA with a state-of-the-art technique to solve the type-1 problem.

3.4. Overall Structure of the Genetic Algorithm

Algorithm 2 provides an outline of the proposed approach. The first step, line 1, is to find an initial feasible solution to the SALBP-2 using Algorithm 1 embedded within a binary search. Then, the population is initialised as IBT-ALBP instances with l random incompatibilities and evaluated using the proposed decoding scheme; see lines 2–4. If an improving solution for the SALBP-2 is found during evaluation, line 4, the best found solution is updated.

After initialization, the algorithm enters its main loop, lines 5–10, where the genetic algorithm operators are performed in parallel. When the total CPU time is reached, the algorithm ends and reports the best found solution for the SALBP-2.

Algorithm 2 Outline of the complete procedure.

```
Input: Parameters $m$, $w$, $t$, $m$, population Size, $mp$, time limit, $l$, and instance definition
 1: Find an initial $c$, see Section 3.2
 2: for $p$:=1...populationSize do                                        ▷ Initialize population
 3:     Create individual and add it to the population, see Section 3.1
 4:     Evaluate the fitness of the individual using Algorithm 1, see Section 3.2
 5: repeat                                                          ▷ Steps performed in parallel
 6:     Select parents, see Section 3.3
 7:     Perform crossover, see Section 3.3
 8:     Apply mutation, see Section 3.3
 9:     Evaluate new individual
10: until time limit is reached
11: Return best found solution for the SALBP-2.
```

4. Computational Experiments

To assess the quality of the proposed method, the algorithm was programmed in C++ and compiled using version 4.6.1 of the GNU GCC compiler. An Intel Core i5 3.2 GHz processor with four cores and 4 GB of RAM running the Linux operating system was used for the experiments. In addition to the proposed method, three other algorithms were implemented for comparison purposes:

1. A reformulation method that uses a binary search to investigate the interval of cycle times as in [8]. Each tentative cycle time is tested using the original BDP [31] for the SALBP-1. The BDP parameters are set to $w = 1600$ and $t = 400$, the largest values tested to solve SALBP-1 instances.
2. A random key-encoded hybrid GA version of the algorithm whose encoding of individuals is similar to the proposal found in [50]. The procedure is also a reformulation method that uses the same approach proposed in Section 3. The GA encodes each individual using a random valued weight for each task of the instance. In [50], these weights were decoded using a station-oriented constructive procedure. In our proposal, the weights substitute the criterion used to reorder tasks in the BDP (a higher weight indicates that the task should appear first in the ordering), which is then used during the fitness evaluation. The modified GA uses the initialisation, crossover, and mutation operators proposed in [50] to handle the weight encoding. In addition to the weight information, each individual also encodes a binary value to determine the direction of the precedence graph given to the BDP. The precedence graph part of each individual uses the operators defined in Section 3.3.
3. A random sampling method. The reformulation approach proposed in Section 3.2 is used. The second phase—the upper-bound search phase—is not conducted using a GA but rather by iteratively creating new IBT-ALBP instances using the procedure proposed in Section 3.1 and solving them using the BDP.

For the remainder of the paper, each algorithm is identified by the following acronyms: BS-BDP represents the binary search exploration of cycle time intervals using the original BDP, RK-GA represents the random key-based genetic algorithm, IBT-RS represents the random sampling method, and IBT-GA represents the genetic algorithm described in Section 3.

Note that each of these methods attempts to verify the efficiency of the components of the final method. BS-BDP studies the efficiency of the original BDP, RK-GA studies the efficiency of the BDP when diversification is introduced without modifying the original instance, IBT-RS studies the behaviour of the proposed method to diversify the solutions provided by the BDP, and, lastly, IBT-GA studies the combined behaviour of each component of the proposed algorithm.

The computational study was carried out using the SALBP-2 benchmark set obtained at www.assembly-line-balancing.de. The instance set is composed of 302 instances, with a number of tasks ranging from 29 to 297. As the performance of each algorithm depends on several parameters, a number of preliminary tests were performed to determine the best performing set of values. After optimisation by hand, the selected values are shown in Table 1, as they were able to produce the best results.

Table 1. Parameters used by the proposed algorithms. Population size, mutation probability (mp), number of incompatibilities introduced on the original instance (l), window width (w), maximum number of transitions (t), and time limit are given.

Population Size = 100	$w = 500$	$l = 20$
Time limit = 3600 s	$mp = 0.03$	$t = 50$

Algorithm BS-BDP, as a deterministic algorithm, was run once without time limit. Alternatively, the rest of the algorithms were run 10 times for each of the 302 instances of the set, with a CPU time limit of 3600 s. Tables 2 and 3 report the results obtained by each of the algorithms.

Table 2 presents a summary of the results for each algorithm and the combined results of any previous method as reported in the benchmark set (row Literature). Two values are reported: the first value (column # Best) is the number of instances in which the best-known solution is obtained by the method; the second value (column # Improves) reports the number of instances in which the algorithm improved the previously best-known solution.

The numbers of best-known and improved solutions demonstrate the quality of the proposed hybrid algorithm and the relative importance of each component of the algorithm. Note that the number of best solutions reported in the literature is equal to 294 instances (out of 302), meaning that the proposed algorithm has improved eight of the previously best-known solutions (see Appendix A for a list of the instances and their improved solutions).

Table 2. Results of the procedures applied to the instance set (composed of 302 instances). For each procedure, the number of instances in which the algorithm provides the best solution (column #Best) or improves upon the best-known solution in the literature (column #Improves) are reported.

Procedure	**# Best**	**# Improves**
Literature	294	
BS-BDP	293	1
RK-GA	296	6
RS-BDP	296	5
IBT-GA	301	8

Table 3. Individual results of the algorithms for the hard instances of the reference set (instances in which the best solution of any of the proposed methods in any of its executions differs from the previous best-known solution in the literature). For each instance, defined by graph name and number of stations (m), the solutions found in the literature (Lit), and the solutions provided by each of the proposed methods are reported. The minimum and average value out of 10 executions are reported for the RK-GA, RS-BDP, and IBT-GA methods. Best-known solutions are indicated in boldface.

		BS-	RK-GA (3600 s)		RS-BDP (3600 s)		IBT-GA (3600 s)	
Graph(m)	Lit.	BDP	Min.	Avg.	Min.	Avg.	Min.	Avg.
Arcus2(19)	7922	7922	**7921**	7921.5	**7921**	7921.5	**7921**	7921.3
Arcus2(20)	7524	**7523**	**7523**	7523.2	**7523**	**7523**	**7523**	7523.1
Arcus2(21)	7187	7186	**7184**	7185.4	**7184**	7185.5	**7184**	7185.1
Arcus2(22)	6856	6856	6859	6859	6858	6858	**6850**	6854.1
Arcus2(23)	6560	6560	**6559**	6564	6560	6562.7	**6559**	6561.4
Arcus2(24)	6282	6290	6286	6295	6286	6292.2	**6280**	6280.6
Arcus2(25)	6101	6118	6108	6110.1	6105	6109.6	**6096**	6096.5
Arcus2(26)	5855	5860	5864	5867.4	5854	5857.4	**5851**	5851.5
Mukherje(20)	**220**	**220**	221	221	221	221	221	221
Barthol2(50)	**85**	86	**85**	**85**	**85**	85.7	**85**	85.5

The results of the BS-BDP confirm the applicability of the reformulation methods to efficiently solve the SALBP-2. BS-BDP is able to obtain a similar number of best-known solutions as the combination of any previous method reported in the literature, and it is able to find a new best-known solution. The results also demonstrate the limitations faced by this method, as discussed in Section 1. While the BDP was able to obtain the best-known solution for 268 instances (out of 269) for the SALBP-1 set [31], it does not obtain the best solution for nine instances of the SALBP-2 set. In addition, note that the parameters ($w = 1600$, $t = 400$) used in the computational experience were chosen to obtain good-quality solutions, disregarding possible high run times, as, for some instances, the BS-BDP requires several hours to obtain the reported solution.

Combining the BDP with a source of diversification makes it possible to partially overcome the perceived shortcoming. RK-GA diversifies the BDP by studying alternative task orderings rather than increasing the width and number of transitions of the DP. The hybrid outperforms the basic BDP and is able to improve the previously best-known solution for six of the instances. This improvement shows the importance of diversifying the states constructed during the enumerate step of the BDP, as opposed to the intensification provided by increasing the value of its parameters. Combining the BDP with a source of diversification makes it possible to partially overcome the perceived shortcoming. RK-GA diversifies the BDP by studying alternative task orderings rather than increasing the width and number of transitions of the DP. The hybrid outperforms the basic BDP and is able to improve the previously best-known solution for six of the instances. This improvement shows the importance of diversifying the states constructed during the enumerate step of the BDP, as opposed to the intensification provided by increasing the value of its parameters.

The results of RK-GA are similar to the results of RS-BDP even if no algorithm directs the introduction of diversity. This shows the efficiency of diversifying the search by constraining the solution space using modified IBT-ALBP instances in contrast to altering the behaviour of the BDP.

The final procedure, IBT-GA, outperforms all of the previously discussed methods, and clearly improves the results from the literature, providing the best-known solution for 301 out of the 302 instances in the benchmark set and providing eight new best-known solutions.

Table 3 provides individual results for the instances in which the best solution of any of the proposed methods in any of its executions differs from the previous best-known solution in the literature. For each instance, denoted by the graph name and the number of stations, the solution found in the literature (column Lit.) and the solutions provided by the proposed algorithms (BS-BDP, RK-GA, RS-BDP, and IBT-GA) are reported. Minimum, average, and maximum values (columns min., av., and max., respectively) for the ten executions of algorithms RK-GA, RS-BDP, and IBT-GA are provided. Best-known solutions are indicated in boldface.

The results for the individual instances also show the quality of the IBT-GA. The average solution outperforms the results from the literature, and even its worst-case performance is comparable to the best previously known solutions. The results lead us to conclude that IBT-GA outperforms previously reported approaches for the problem if low run times (solutions obtained within seconds) is not a critical concern.

The results for individual instances also show that no single method dominates the other methods. Although the IBT-GA provides most of the best-known solutions, there is a case, the Mukherje graph with 20 stations, in which the original BDP algorithm outperforms the solutions given by the procedure.

The main weakness of the proposed method is its run time, but good results are also obtained with smaller time limits. For a time limit of 600 s, IBT-GA was able to obtain the best solution for 291 instances in every one of the 10 runs. In addition, note that 261 instances were optimally solved during phase 1 of the proposed method. Overall, we can conclude the following: (1) the considerable run times are only required for the hardest instances of the set, and (2) the proposed GA offers a slow convergence rate and a degree of diversification, which are desirable characteristics to hybridise with the intensification provided by the BDP.

5. Discussion and Conclusions

In this work, we have proposed a hybrid genetic algorithm for the simple assembly line balancing problem with a fixed number of stations, the SALBP-2. The experimental evaluation of the algorithm shows the quality of the proposed method, which is capable of providing eight new best-known solutions for the benchmark set used in the literature.

The hybrid GA uses a Dynamic Programming-based heuristic, the BDP procedure, to evaluate the individuals. These individuals represent instances of a modified problem rather than solutions to the original SALPB-2. The modified problem corresponds to the assembly line balancing problem with incompatibilities between tasks. This study also contributes to the state of the art of this problem by developing a new lower bound and several reduction rules for the balancing problem with incompatibilities between tasks.

While the proposed operators are simple, the GA part of the hybrid provides enough diversification to drive the BDP away from local optimality.

The main weakness of the proposed algorithm is its run time; thus, it is only applicable if enough time is available or the user accepts a possible deterioration in the quality of the solution. However, note that balancing assembly lines is not a time-critical application, as it is not a daily decision, and thus the run time required by the proposed method is usually available. Moreover, the solutions obtained with reduced run times are still competitive with those found in the literature.

We would also like to point out the applicability of similar-based methods for other line balancing problems, including multicriteria problems, where a high-performing constructive method may be available.

Author Contributions: Conceptualization, E.Á.-M. and J.P.; methodology, E.Á.-M. and J.P.; software, J.P. and M.V.; validation, J.P., H.T.-M. and M.V.; formal analysis, E.Á.-M. and J.P.; investigation, E.Á.-M. and J.P.; resources, J.P.; data curation, E.Á.-M.; writing—original draft preparation, J.P., H.T.-M., and M.V.; writing—review and editing, J.P.; visualization, E.Á.-M., J.P., H.T.-M. and M.V.; supervision, E.Á.-M., J.P., H.T.-M. and M.V.; funding acquisition, E.Á.-M. and J.P. All authors have read and agreed to the published version of the manuscript.

Funding: E. Álvarez-Miranda acknowledges the support of Chilean National Agency for Research and Development (ANID) through the grant FONDECYT N.1191624 "Assembly line balancing for industry 4.0" and through the Complex Engineering Systems Institute ANID PIA/BASAL AFB180003. J.P. acknowledges the support of ANID through the grant FONDECYT No.1191624 "Assembly line balancing for industry 4.0".

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Dataset available at http://www.assembly-line-balancing.de (accessed on 2 September 2021) or upon request to the authors.

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

Appendix A. Improved Solutions

List of new best-known solutions: The instances are identified by graph name and number of workstations m. The best cycle time c is reported, and the task assignments are reported as a list of n integers in which the ith position indicates the workstation assignment of task i.

Graph: Arcus2, m = 19, c = 7921. Task Assignment = {1, 1, 1, 1, 1, 8, 4, 4, 1, 2, 2, 2, 5, 3, 2, 2, 5, 2, 6, 7, 5, 6, 3, 8, 4, 4, 2, 6, 3, 10, 6, 3, 14, 5, 3, 7, 4, 11, 18, 18, 8, 6, 4, 12, 7, 15, 7, 7, 11, 18, 8, 19, 14, 8, 9, 11, 9, 7, 14, 8, 9, 10, 11, 12, 9, 7, 15, 11, 10, 9, 10, 12, 12, 12, 13, 13, 13, 14, 16, 15, 15, 15, 15, 14, 16, 15, 15, 19, 16, 19, 16, 17, 16, 17, 17, 17, 18, 17, 19, 19, 18, 19, 17, 17, 19, 19, 19, 17, 19, 18,19}.

Graph: Arcus2, m = 20, c = 7523. Task Assignment = {1, 1, 1, 1, 3, 2, 1, 2, 1, 2, 2, 3, 6, 3, 8, 3, 3, 3, 5, 7, 9, 3, 4, 11, 3, 18, 6, 6, 4, 8, 10, 7, 5, 4, 7, 6, 4, 9, 19, 20, 7, 5, 8, 13, 8, 9, 6, 8, 10, 16, 10, 11, 14, 10, 8, 10, 6, 8, 12, 7, 8, 9, 20, 12, 11, 8, 13, 20, 13, 11, 11, 12, 13, 13, 14, 13, 14, 15, 14, 15, 14, 18, 15, 15, 16, 20, 14, 16, 17, 16, 16, 16, 16, 16, 17, 18, 17, 17, 20, 19, 18, 18, 19, 20, 18, 18, 19, 19, 20, 19, 20}.

Graph: Arcus2, m = 21, c = 7184. Task Assignment = {1, 1, 1, 1, 1, 2, 1, 11, 5, 2, 2, 2, 16, 3, 13, 4, 2, 4, 2, 5, 3, 3, 4, 5, 4, 21, 4, 4, 6, 2, 3, 6, 5, 5, 4, 8, 6, 2, 6, 7, 9, 7, 4, 12, 7, 8, 8, 11, 15, 15, 18, 10, 11, 8, 8, 11, 9, 10, 14, 12, 9, 15, 13, 12, 17, 21, 14, 20, 9, 10, 16, 12, 10, 13, 11, 14, 17, 13, 14, 18, 15, 21, 17, 16, 15, 19, 15, 19, 17, 16, 18, 18, 19, 18, 19, 20, 20, 19, 21, 21, 20, 20, 20, 21, 20, 20, 20, 20, 21, 20, 21}.

Graph: Arcus2, m = 22, c = 6850. Task Assignment = {1, 1, 1, 1, 1, 4, 1, 2, 2, 2, 3, 2, 7, 3, 3, 6, 3, 4, 5, 7, 21, 6, 4, 10, 5, 6, 6, 5, 4, 6, 16, 4, 21, 6, 6, 5, 8, 6, 22, 22, 5, 7, 6, 7, 8, 9, 8, 10, 12, 11, 9, 15, 19, 10, 8, 9, 8, 16, 10, 11, 10, 14, 9, 12, 12, 16, 11, 11, 10, 11, 17, 13, 12, 13, 12, 13, 15, 15, 13, 14, 17, 14, 15, 16, 14, 17, 18, 17, 17, 18, 18, 19, 18, 19, 19, 19, 20, 20, 20, 21, 20, 20, 20, 19, 20, 21, 21, 21, 22, 22, 22}.

Graph: Arcus2, m = 23, c = 6559. Task Assignment = {1, 1, 1, 1, 6, 2, 1, 4, 8, 2, 2, 2, 7, 3, 4, 2, 2, 2, 3, 9, 5, 10, 3, 4, 5, 6, 4, 2, 4, 3, 10, 9, 4, 6, 5, 2, 6, 3, 23, 23, 10, 7, 7, 6, 6, 12, 8, 10, 7, 11, 9, 18, 12, 10, 8, 9, 11, 17, 11, 14, 11, 17, 22, 13, 22, 19, 13, 18, 11, 12, 18, 13, 12, 14, 12, 15, 16, 14, 13, 15, 16, 16, 18, 14, 15, 16, 17, 18, 15, 17, 19, 20, 19, 19, 20, 21, 20, 21, 21, 21, 21, 20, 21, 21, 22, 20, 22, 21, 20, 21, 23}.

Graph: Arcus2, m = 24, c = 6280. Task Assignment = {1, 1, 1, 1, 3, 2, 7, 9, 4, 2, 2, 2, 7, 3, 14, 3, 5, 7, 6, 11, 22, 4, 3, 6, 4, 17, 4, 5, 8, 6, 11, 5, 4, 5, 10, 6, 8, 10, 16, 23, 10, 6, 10, 13, 9, 16, 7, 11, 10, 13, 17, 19, 12, 8, 7, 12, 19, 8, 8, 17, 10, 9, 15, 13, 22, 8, 11, 23, 10, 11, 13, 14, 12, 14, 14, 15, 18, 16, 17, 19, 17, 24, 18, 19, 20, 24, 18, 24, 24, 19, 20, 20, 21, 21, 21, 22, 22, 21, 23, 23, 22, 24, 23, 23, 24, 24, 24, 23, 24, 23, 24}.

Graph: Arcus2, m = 25, c = 6096. Task Assignment = {1, 1, 1, 2, 8, 4, 2, 5, 4, 2, 2, 3, 9, 3, 3, 3, 4, 5, 5, 7, 15, 4, 5, 6, 4, 22, 6, 11, 6, 6, 6, 9, 22, 6, 6, 13, 8, 8, 23, 25, 11, 7, 6, 17, 9, 12, 8, 9, 11, 19, 12, 19, 12, 9, 10, 10, 9, 13, 11, 10, 12, 13, 24, 12, 21, 16, 12, 11, 13, 14, 19, 14, 15, 14, 15, 16, 18, 16, 16, 17, 17, 24, 19, 17, 18, 18, 19, 18, 18, 20, 20, 21, 20, 21, 21, 24, 23, 22, 25, 22, 24, 24, 22, 24, 25, 24, 24, 23, 25, 23, 25}.

Graph: Arcus2, m = 26, c = 5851. Task Assignment = {1, 1, 1, 2, 2, 2, 2, 2, 5, 3, 3, 3, 3, 4, 23, 6, 3, 4, 5, 7, 17, 5, 9, 5, 4, 23, 6, 4, 5, 7, 13, 11, 25, 5, 9, 4, 5, 8, 24, 25, 11, 6, 9, 6, 6, 9, 7, 10, 10, 6, 11, 14, 15, 10, 7, 8, 8, 19, 12, 14, 9, 10, 25, 12, 11, 23, 13, 14, 12, 13, 18, 15, 14, 16, 14, 16, 15, 16, 16, 19, 17, 26, 18, 19, 18, 22, 17, 21, 22, 17, 20, 21, 20, 21, 21, 22, 22, 22, 22, 26, 26, 23, 24, 23, 26, 25, 24, 24, 26, 24, 26}.

References

1. Salveson, M.E. The assembly line balancing problem. *J. Ind. Eng.* **1954**, *6*, 18–25.
2. Battaïa, O.; Dolgui, A. A taxonomy of line balancing problems and their solution approaches. *Int. J. Prod. Econ.* **2013**, *142*, 259–277. [CrossRef]
3. Scholl, A.; Becker, C. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *Eur. J. Oper. Res.* **2006**, *168*, 666–693. [CrossRef]
4. Becker, C.; Scholl, A. A survey on problems and methods in generalized assembly line balancing. *Eur. J. Oper. Res.* **2006**, *168*, 694–715. [CrossRef]
5. Baybars, I. A survey of exact algorithms for the simple assembly line balancing problem. *Manag. Sci.* **1986**, 32, 909–932. [CrossRef]
6. Li, Y. The type-ii assembly line rebalancing problem considering stochastic task learning. *Int. J. Prod. Res.* **2017**, *55*, 7334–7355. [CrossRef]
7. Sancı, E.; Azizoğlu, M. Rebalancing the assembly lines: Exact solution approaches. *Int. J. Prod. Res.* **2017**, *55*, 5991–6010. [CrossRef]
8. Scholl, A. *Balancing and Sequencing of Assembly Lines*, 2nd ed.; Springer-Verlag: Berlin/Heidelberg, Germany, 1999.
9. Scholl, A.; Voss, S. Simple assembly line balancing—Heuristic approaches. *J. Heur.* **1996**, *2*, 217–244. [CrossRef]
10. Ugurdag, H.F.; Rachamadugu, R.; Papachristou, C.A. Designing paced assembly lines with fixed number of stations. *Eur. J. Oper. Res.* **1997**, *102*, 488–501. [CrossRef]
11. Roshani, A.; Paolucci, M.; Giglio, D.; Tonelli, F. A hybrid adaptive variable neighbourhood search approach for multi-sided assembly line balancing problem to minimise the cycle time. *Int. J. Prod. Res.* **2021**, *59*, 3696–3721. [CrossRef]
12. Lopes, T.C.; Michels, A.S.; Lüders, R.; Magatão, L. A simheuristic approach for throughput maximization of asynchronous buffered stochastic mixed-model assembly lines. *Coput. Oper. Res.* **2020**, *115*, 104863.
13. Klein, R.; Scholl, A. Maximizing the production rate in simple assembly line balancing—A branch and bound procedure. *Eur. J. Oper. Res.* **1996**, *91*, 367–385. [CrossRef]
14. Pınarbaşı, M.; Alakaş, H.M. Balancing stochastic type-II assembly lines: Chance-constrained mixed integer and constraint programming models. *Eng. Opt.* **2020**, *52*, 2146–2163. [CrossRef]
15. Abidin Çil, Z.; Kizilay, D. Constraint programming model for multi-manned assembly line balancing problem. *Comput. Oper. Res.* **2020**, *124*, 105069. [CrossRef]
16. Kilincci, O. A Petri net-based heuristic for simple assembly line balancing problem of type 2. *Int. J. Adv. Manuf. Technol.* **2010**, *46*, 329–338. [CrossRef]
17. Nearchou, A.C. Balancing large assembly lines by a new heuristic based on differential evolution method. *Int. J. Adv. Manuf. Technol.* **2007**, *34*, 1016–1029. [CrossRef]
18. Blum, C. Iterative beam search for simple assembly line balancing with a fixed number of work stations. *Stat. Oper. Res. Trans.* **2011**, *35*, 145–164.
19. Li, Z.; Janardhanan, M.N.; Ponnambalam, S.G. Cost-oriented robotic assembly line balancing problem with setup times: Multi-objective algorithms. *J. Intell. Manuf.* **2021**, *32*, 989–1007. [CrossRef]
20. Zhang, H.; Yan, Q.; Liu, Y.; Jiang, Z. An integer-coded differential evolution algorithm for simple assembly line balancing problem of type 2. *Assem. Autom.* **2016**, *36*, 246–261. [CrossRef]
21. Fang, Y.; Ming, H.; Li, M.; Liu, Q.; Pham, D.T. Multi-objective evolutionary simulated annealing optimisation for mixed-model multi-robotic disassembly line balancing with interval processing time. *Int. J. Prod. Res.* **2020**, *58*, 846–862. [CrossRef]
22. Meng, K.; Tang, Q.; Zhang, Z.; Qian, X. An Improved Lexicographical Whale Optimization Algorithm for the Type-II Assembly Line Balancing Problem Considering Preventive Maintenance Scenarios. *IEEE Access* **2020**, *8*, 30421–30435. [CrossRef]
23. Cerqueus, A.; Delorme, X. A branch-and-bound method for the bi-objective simple line assembly balancing problem. *Int. J. Prod. Res.* **2019**, *57*, 5640–5659. [CrossRef]
24. Li, Y.; Wang, H.; Yang, Z. Type II assembly line balancing problem with multi-operators. *Neural Comput. Appl.* **2019**, *31*, 347–357.
25. Cao, Y.; Li, Y.; Liu, Q.; Zhang, J. An Optimization Model for Assembly Line Balancing Problem with Uncertain Cycle Time. *Math. Probl. Eng.* **2020**, *2020*, 2785278. [CrossRef]
26. Bukchin, Y.; Raviv, T. Constraint programming for solving various assembly line balancing problems. *Omega* **2018**, *78*, 57–68. [CrossRef]
27. Borba, L.; Ritt, M.; Miralles, C. Exact and heuristic methods for solving the Robotic Assembly Line Balancing Problem. *Eur. J. Oper. Res.* **2018**, *270*, 146–156. [CrossRef]
28. 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]
29. Pinarbasi, M.; Alakas, H.M.; Yuzukirmizi, M. A constraint programming approach to type-2 assembly line balancing problem with assignment restrictions. *Assem. Autom.* **2019**, *39*, 813–826. [CrossRef]
30. Goldberg, D. *Genetic Algorithms in Search, Optimization, and Machine Learning*; Addison-Wesley: Reading, MA, USA, 1989.
31. Bautista, J.; Pereira, J. A dynamic programming based heuristic for the assembly line balancing problem. *Eur. J. Oper. Res.* **2009**, *194*, 787–794. [CrossRef]

32. Scholl, A.; Boysen, N.; Fliedner, M. ABSALOM: Balancing assembly lines with assignment restrictions. *Eur. J. Oper. Res.* **2010**, *200*, 688–701. [CrossRef]
33. Boysen, N.; Fliedner, M. A versatile algorithm for assembly line balancing. *Eur. J. Oper. Res.* **2008**, *184*, 39–55. [CrossRef]
34. Vilarinho, P.M.; Simaria, A.S. A two-stage heuristic method for balancing mixed- model assembly lines with parallel stations. *Int. J. Prod. Res.* **2002**, *40*, 1405–1420. [CrossRef]
35. Vilarinho, P.M.; Simaria, A.S. ANTBAL: An ant colony optimization algorithm for balancing mixed-model assembly lines with parallel stations. *Int. J. Prod. Res.* **2006**, *44*, 291–303. [CrossRef]
36. Bautista, J.; Pereira, J. Ant algorithms for assembly line balancing. *Lect. Notes Comput. Sci.* **2002**, *2463*, 65–75.
37. Lapierre, S.D.; Ruiz, A.B. Balancing assembly lines: An industrial case study. *J. Oper. Res. Soc.* **2004**, *55*, 589–597. [CrossRef]
38. Johnson, R.V. Optimally balancing large assembly lines with 'fable'. *Manag. Sci.* **1988**, *34*, 240–253. [CrossRef]
39. Tarjan, R.E. Depth-first search and linear graph algorithms. *SIAM J. Comput.* **1972**, *1*, 146–160. [CrossRef]
40. Martello, S.; Toth, P. *Knapsack Problems: Algorithms and Computer Implementations*; John Wiley & Sons: Hoboken, NJ, USA, 1990.
41. Jackson, J.R. A computing procedure for a line balancing problem. *Manag. Sci.* **1956**, *2*, 261–271. [CrossRef]
42. Bautista, J.; Pereira, J. Procedures for the Time and Space constrained Assembly Line Balancing Problem. *Eur. J. Oper. Res.* **2011**, *212*, 473–481. [CrossRef]
43. Hoffmann, T.R. Assembly line balancing with a precedence matrix. *Manag. Sci.* **1963**, *9*, 551–562. [CrossRef]
44. Holland, J.H. *Adaptation in Natural and Artificial Systems*; The University of Michigan Press: Ann Arbor, MI, USA, 1975.
45. Tasan, S.O.; Tunali, S. A review of current applications of genetic algorithms in assembly line balancing. *J. Intell. Manuf.* **2008**, *19*, 49–69. [CrossRef]
46. Kim, Y.K.; Kim, Y.; Kim, Y.J. Two-sided assembly line balancing: A genetic algorithm approach. *Prod. Plan. Control* **2000**, *11*, 44–53. [CrossRef]
47. Falkenauer, E. A hybrid grouping genetic algorithm for bin packing. *J. Heur.* **1996**, *2*, 5–30. [CrossRef]
48. Rekiek, B.; de Lit, P.; Pellichero, F.; Eglise, T.; Fouda, P.; Falkenauer, E.; Delchambre, A.A multiple objective grouping genetic algorithm for assembly line balancing. *J. Intell. Manuf.* **2001**, *12*, 467–485. [CrossRef]
49. Sabuncuoglu, I.; Erel, E.; Tanyer, M. Assembly line balancing using genetic algorithms. *J. Intell. Manuf.* **2000**, *11*, 295–310. [CrossRef]
50. Gonçalves, J.F.; Almeida, J.R. A hybrid genetic algorithm for assembly line balancing. *J. Heur.* **2002**, *8*, 629–642. [CrossRef]
51. 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]
52. Mutlu, O.; Polat, O.; Supciller, A.A. An iterative genetic algorithm for the assembly line worker assignment and balancing problem of type-II. *Comput. Oper. Res.* **2013**, *40*, 418–426. [CrossRef]
53. Bautista, J.; Suárez, R.; Mateo, M.; Companys, R. Local search heuristics for the assembly line balancing problem with incompatibilities between tasks. In Proceedings of the 2000 IEEE International Conference on Robotics and Automation, San Francisco, CA, USA, 24–28 April 2000; pp. 2404–2409.
54. Simaria, A.S.; Vilarinho, P.M. A genetic algorithm based approach to the mixed-model assembly line balancing problem of type II. *Comput. Ind. Eng.* **2004**, *47*, 391–407. [CrossRef]
55. Luque, G.; Alba, E.; Dorronsoro, B. Parallel Genetic Algorithms, In *Parallel Metaheuristics*; Alba, E., Ed.; Wiley: Hoboken, NJ, USA, 2005; pp. 107–125.