Solving the accessibility windows assembly line problem level 1 and variant 1 (AWALBP-L1-1) with precedence constraints

Alberto GARCÍA-VILLORIA[a,*], Albert COROMINAS[a], Adrià NADAL[b] and Rafael PASTOR[a]

[a] Institute of Industrial and Control Engineering (IOC), Universitat Politècnica de Catalunya (UPC), Spain

[b] Volkswagen-Audi España S.A., Spain

{alberto.garcia-villoria / albert.corominas / rafael.pastor}@upc.edu, adria.nadal.sola@gmail.com

Abstract. Assembly line balancing problems (ALBPs) are among the most studied combinatorial optimization problems due to their relevance in many production systems. In particular, the accessibility windows ALBP (AWALBP) may arise when the workpieces are larger than the workstations, which implies that at a given instant the workstations have access to only a portion of the workpieces. Thus, the cycle is split into forward steps and stationary stages. The workpieces advance during the forward steps and the tasks are processed during the stationary stages. Several studies have dealt with the AWALBP assuming that there are no precedence relationships between tasks. However, this assumption is not always appropriate. In this work we solve the first level of AWALBP (AWALBP-L1) considering the existence of precedence relationships. Specifically, this work deals with variant 1 (AWALBP-L1-1), in which each task can be performed at only one workstation and, therefore, only the stationary stages and the starting instants in which the tasks are performed have to be decided. We design a solution procedure that includes pre-processing procedures, a matheuristic and a mixed integer linear programming model. An extensive computational experiment is carried out to evaluate its performance.

Keywords: manufacturing; assembly line balancing; accessibility windows; matheuristic procedure

1. Introduction

Assembly lines are very common in many mass production systems and they give rise to a variety of problems. In particular, the family of problems known as assembly line balancing problems (ALBPs) emerges. The core of ALBPs consists in assigning tasks to an ordered sequence of workstations so that some constraints are satisfied and one or more efficiency objectives are optimised. Over the past six decades, different types of ALBPs have been studied in the literature; see, for instance, the last surveys: Becker and Scholl (2006), Scholl and Becker (2006), Boysen *et al*. (2007, 2008) and Battaïa and Dolgui (2013).

[*] Corresponding author: Alberto García-Villoria, Institute of Industrial and Control Engineering (IOC), Av. Diagonal 647 (Edif. ETSEIB), 11th floor, 08028 Barcelona, Spain; tel.: +34 93 4010724; e-mail: alberto.garcia-villoria@upc.edu

Even the easiest ALBP, called simple ALBP (SALBP) is NP-hard (e.g., Baybars, 1986). SALBP is defined with the following assumptions (Baybars, 1986): 1) a task cannot be split among workstations; 2) there are precedence relationships between tasks; 3) all tasks must be processed; 4) the processing times of the tasks are additive, workstation-independent, sequence-independent and known with certainty; 5) all workstations have the same associated costs; 6) any task can be performed at any workstation; 7) the line is serial and without feeder or parallel subassembly lines; 8) the line is designed for a unique model of a single product. A branch and bound based method has recently been designed (Sewell and Jacobson, 2012) and improved (Morrison *et al*., 2014), which is able to solve optimally and very quickly large instances of SALBP (although heuristic techniques may be needed to solve bigger instances).

On the other hand, SALBP is more an academic abstraction than a real-world problem. Thus, in the last years researchers have intensified their efforts, studying ALBPs with additional characteristics of real systems. For instance, among many others: task times depending on the sequence (e.g., Capacho *et al*., 2009), setup times between tasks (e.g., Akpinar *et al*., 2017), space constraints (e.g., Bautista and Pereira, 2011), constrained resources (e.g., Quyen *et al*., 2017), ergonomics considerations (e.g., Otto and Battaïa, 2017), mixed-models (e.g., Zelter *et al*., 2017), U-shaped lines (e.g., Fathi *et al*., 2017), disassembly lines (Bentaha *et al*., 2015), two-sided assembly lines (e.g., Abdullah Make *et al*., 2017), robotic assembly lines (e.g., Borba *et al*., 2018), uncertainty on task times (e.g., Krishnan *et al.*, 2016), task times dependent on the workers (e.g., Moreira *et al*., 2015) and task times under a learning effect (Koltai and Kalló, 2017).

In all the aforementioned works, it is assumed (explicitly or implicitly) that, at any moment, each workstation has full access to one workpiece and tasks are performed on each workpiece at only one workstation. However, in some production systems as, for instance, printed circuit boards (PCBs) manufacturing, this assumption may be inappropriate since the size of the workpiece is large relative to the dimensions of the workstations and the *accessibility windows* of the workstations are smaller than the workpiece (Müller-Hannemann and Weihe, 2006). Therefore, at a given moment, each workpiece on the line may be accessible from several workstations and each workstation may perform tasks on one or two consecutive workpieces (e.g., Figure 1). This characteristic gives rise to the family of problems called accessibility windows ALBP (AWALBP) (Calleja *et al*., 2013).

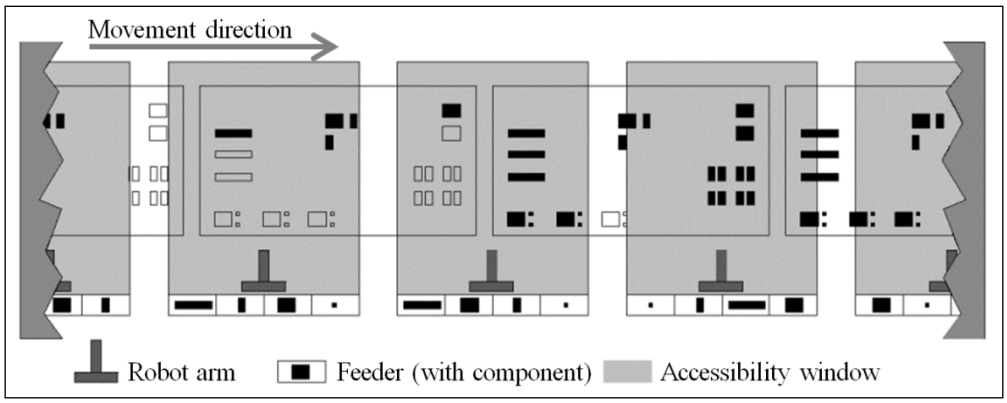

Figure 1. Example of an assembly line with three workstations with accessibility windows (grey areas). Reprinted from Calleja *et al*. (2016) with permission

In the AWALBP, production cycles are split into *stationary stages* and *forward steps*. The line becomes motionless during each stationary stage so the workstations can perform the tasks assigned to them when these are placed within the corresponding accessibility window. Between two stationary stages there is one forward step, in which the line moves forward. This movement means that the workstations can access new tasks. Figure 2 illustrates a line with four workstations and a cycle consisting of three stationary stages (and, therefore, three forward steps). The movement length of forward step σ ($\sigma = 1,...,3$) is indicated as δ_σ, and x indicates the position of the workpieces at the start of the cycle, which is the distance between the right border of the first workpiece on the line and the left border of the accessibility windows of the first workstation.

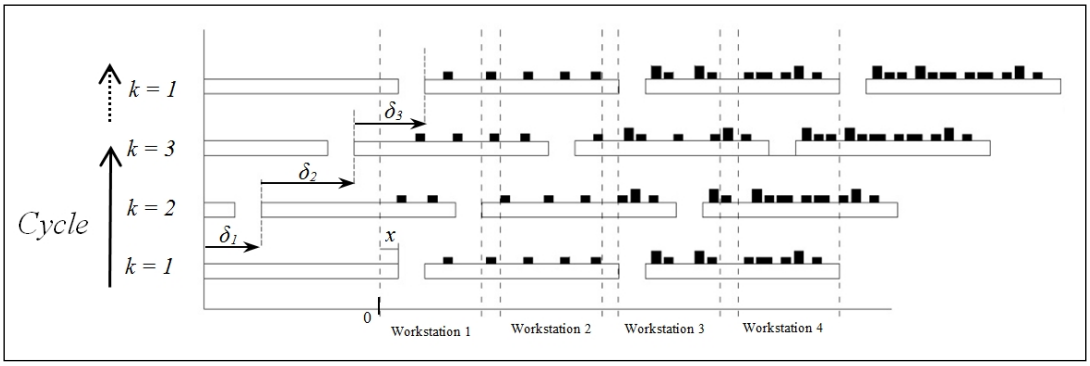

Figure 2. Example of a cycle separated in three stationary stages. Reprinted from Calleja *et al*. (2016) with permission

Calleja *et al*. (2013) classify the tactical and operational decisions in the AWALBP into the following four levels (L1 to L4): L1) assignment of tasks to workstations and stationary stages; L2) *movement scheme*, which is defined by the initial shift x, the number of stationary stages and the lengths of the forward steps; L3) configuration of the workstations (types of components, feeders, toolbits, etc.); and L4) configuration of the line (number of workstations, technology of the line, etc.). According to these four levels, four types of the AWALBP are

defined: AWALBP-L1 to AWALBP-L4. Each variant consists of the solution of its level and the precedent ones and assumes that the other levels have been solved. For instance, AWALBP-L2 consists of the assignment of tasks to workstations and stationary stages and the definition of the movement scheme, given the configurations of the workstations and the line. Fleszar (2017) extends the classification and discerns two variants: variant 1, in which each task can be performed only at one workstation (e.g., AWALBP-L2-1), and variant M, in which tasks can be performed at more than one workstation (e.g., AWALBP-L2-M). Variant 1 occurs, for instance, in some assembly lines of PCBs (see Müller-Hannemann and Weihe, 2006).

In comparison with other ALBPs, there are few studies on the AWALBP. Some PhD and diploma theses (Martin, 2002; Gaudlitz, 2004; Tazari, 2006; Stille, 2008) were developed in collaboration with Philips/Assembléon to study specific cases of PCB automated assembly lines. More general studies on the AWALBP focus on levels L1 and L2. AWALBP-L1-1 is solved heuristically in Müller-Hannemann and Weihe (2006) and optimally with mixed integer linear programming (MILP) in Calleja *et al*. (2014). García-Villoria *et al*. (2015) propose heuristics and simulated annealing procedures for AWALBP-L1-M considering that the processing times of the tasks depend on the workstations. Two MILP models are proposed in Calleja *et al*. (2013) and metaheuristics and matheuristics procedures in Calleja *et al*. (2014, 2016) to solve AWALBP-L2-1. Fleszar (2017) proposes two MILP models for solving AWALBP-L2-1 and AWALBP-L2-M, respectively, and points out that AWALBP-L2-M is much more difficult to solve than AWALBP-L2-1.

All the aforementioned general studies assume that there are no precedence relationships between tasks. However, there are cases in which precedence relationships appear. For instance, in the manufacturing of PCBs, sockets must be mounted before placing other components. In order to start to fill this gap in the literature, in this work we deal with AWALBP-L1-1 including precedence relationships.

In the classical ALBP literature, the precedence relationship between predecessor task j and successor task k can be ensured when task k is not assigned at a workstation previous to the workstation to which task j is assigned. In the AWALBP this is not necessarily true and precedence relationships may or may not be satisfied regardless of the workstation at which the tasks are performed. Moreover, if tasks j and k are performed at different workstations, then an idle time may appear at the workstation in which task k is performed, because task k cannot start until task j is completed (recall that different workstations can perform tasks on the same workpiece during the same cycle). Thus, the order in which the tasks are processed at each

workstation and stationary stage influences the productivity of the line. These characteristics complicate substantially the formulation and solution of the AWALBP.

In this paper, we design two pre-processing procedures, heuristic, metaheuristic and matheuristic procedures, and a MILP model for the AWALBP-L1-1 with precedence relationships. The scheme of the procedure that we propose to solve the problem is as follows. First, we apply pre-processing and, afterwards, a matheuristic procedure which combines MILP, heuristic and metaheuristic techniques. Additionally, when a feasible solution is found but its optimality is not proven, we try to solve a MILP model during limited computing time.

The remainder of the paper is organised as follows: First, Section 2 describes the problem. Section 3 proposes two pre-processing procedures for the problem. The MILP model and the matheuristic procedure are explained in Sections 4 and 5, respectively (although in the proposed solution procedure the matheuristic precedes the MILP model, this latter is explained first for the sake of clarity). The results of the computational experiment that was carried out are shown and analysed in Section 6. Finally, Section 7 provides the conclusions and some suggestions for future research.

2. Problem statement

The problem dealt with in this study is defined as follows.

The line processes a (potentially infinite) number of identical workpieces. Let b_0 be the length of the workpieces and b ($b \geq b_0$) the sum of b_0 and the gap (if any) between two consecutive workpieces.

There is a unique serial assembly line with nw workstations; let $I = \{1,\ldots,nw\}$ be the set of workstations. The accessibility window of workstation $i \in I$ is determined by the range $[l_i, r_i]$ where $r_i > l_i$ (the accessibility windows concerns only one dimension, the one in which the line moves, see Figures 1 and 2); without loss of generality, it is assumed that $l_1 = 0$. The accessibility windows do not overlap; that is, $l_{i+1} > r_i$. The workpieces are larger than the accessibility window of at least one workstation; that is, $\exists_{i \in I} \mid b_0 > r_i - l_i$.

There are nt tasks that have to be performed on each workpiece fulfilling the precedence relationships; let $J = \{1,\ldots,nt\}$ be the set of tasks. For each task $j \in J$, the unique workstation

w_j in which task j can be performed, its processing time p_j, and the distance a_j to the right border of the workpiece on which task j must be performed, are known.

The cyclical movement of the workpieces is described by a given movement scheme. Recall that a movement scheme is defined by the initial shift of the line (x), the number of forward steps (ns) and the lengths δ_σ of the forward steps ($\sigma \in \{1,\ldots,ns\}$).

The data nomenclature is given in Table 1 and some of them are illustrated in Figure 3.

b_0	Length of the workpiece
b	Length of the workpiece plus the gap between two consecutive workpieces
nt, J	Number and set of tasks, respectively, where $J = \{1,\ldots,nt\}$
p_j	Processing time of task $j \in J$; w.l.o.g. assumed to be integer
w_j	Unique workstation able to process task $j \in J$
a_j	Distance to the right border of the workpiece from the position at which task $j \in J$ is to be performed: $0 \le a_j \le b_0$
PR	Set of precedence relationships between tasks. Task j is an immediate predecessor of task k if and only if $(j,k) \in PR$
nw, I	Number and set of workstations, respectively, where $I = \{1,\ldots,nw\}$
$[l_i, r_i]$	Accessibility window of workstation $i \in I$, where $l_1 = 0$, $l_1 < r_1$ and $r_{i-1} < l_i < r_i$, $i \in I \setminus \{1\}$
x	Initial shift of the line at the start of the cycle, which corresponds to the distance from the right border of the first workpiece on the line with respect to the left limit of workstation 1 ($l_1 = 0$)
ns, S	Number and set of forward steps (or, accordingly, number and set of stationary stages) of the movement scheme, where $S = \{1,\ldots,ns\}$
δ_σ	Length of forward step $\sigma \in S$

Table 1. Data of the problem

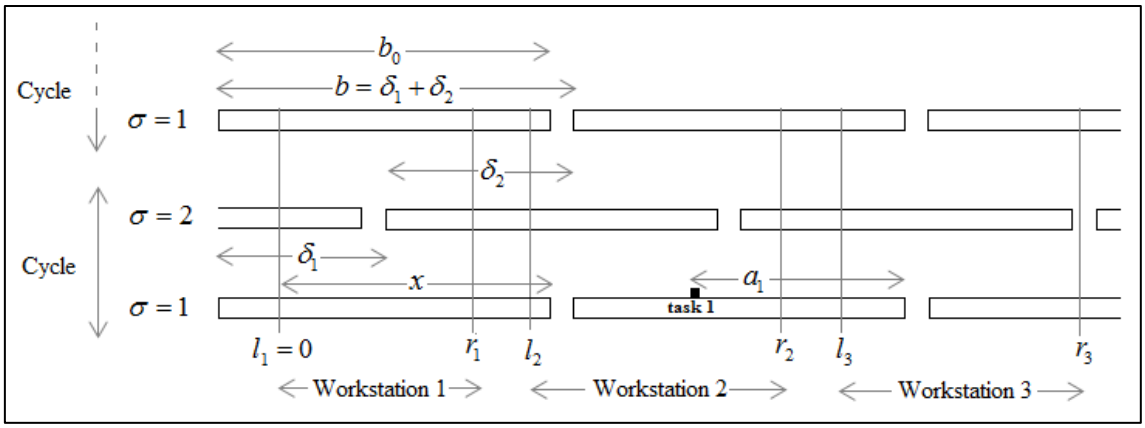

Figure 3. Illustration of some data of the problem with 3 workstations ($nw = 3$) and cycles split into 2 stationary stages and 2 forward steps ($ns = 2$).

And the derived data are given in Table 2.

J_i	Set of tasks to be processed at workstation $i \in I$: $J_i = \{ j \in J : w_j = i \}$
PT_j^I	Set of immediate predecessors of task $j \in J$: $PT_j^I = \{ k \in J : (k, j) \in PR \}$
PT_j^T	Set of all predecessors of task $j \in J$: $PT_j^T = PT_j^I \cup \bigcup_{k \in PT_j^I} PT_k^T$
ST_j^I	Set of immediate successors of task $j \in J$: $ST_j^I = \{ k \in J : (j, k) \in PR \}$
ST_j^T	Set of all successors of task $j \in J$: $ST_j^T = ST_j^I \cup \bigcup_{k \in ST_j^I} ST_k^T$

Table 2. Derived data

A solution of the problem consists of the following two interrelated decisions:

- The assignment of the tasks to the stationary stages in which the tasks are performed. Note that, since there is only one workstation that can perform each task, the assignment of the tasks to workstations is given.

- The scheduling of the tasks; i.e., the instants in which the tasks start to be processed. Note that, in each workstation, there can be idle times between the processing of tasks due to precedence relationships.

The objective is to minimise the cycle time, which in the level L1 (in which the number of stationary steps is fixed) is equivalent to minimising the sum, for all the stationary stages, of the maximum completion times of the tasks performed in each stationary stage. Thus, according to Baybars' nomenclature (Baybars, 1986), AWALBP-L1-1 is a type 2 ALBP.

3. Pre-processing procedures

We have designed two pre-processing procedures for the problem. The first one defines a *task accessibility window* more accurately than the accessibility window of its workstation. The second one identifies the precedence relationships that will be satisfied without imposing them explicitly. These procedures will be helpful for the MILP model (Section 4) and the matheuristic (Section 5).

3.1. Pre-processing rule 1 - Accessibility windows of tasks

The existence of accessibility windows implies that a given task $j \in J$ must be performed within the range $\left[l_{w_j}, r_{w_j}\right]$. Additionally, the existence of precedence relationships between tasks may imply that a given task has to be performed within a narrower range to allow the tasks preceding or succeeding it to be performed within the accessibility windows of their workstations. Additionally, as will be explained later, this range can be set even more accurately if the movement scheme is taken into account.

This idea is illustrated with the following example. Let the pairs of tasks $\{(1,2),(2,3)\} \subseteq PR$, $w_1 = 1$, $w_2 = w_3 = 2$, $r_1 - l_1 = r_2 - l_2 = 10$, $a_1 - a_2 = l_2 - l_1 + 4$ and $a_2 - a_3 = 4$, and suppose that the first position in which task 1 is accessible to its workstation is equal to l_1 (see Figure 4). Thus, we can see that task 2 cannot start to be performed until it is at least in position $l_2 + 4$ (so task 1 can be processed before task 2) and no further than position $l_2 + 6$ (so task 3 can be processed later than task 2).

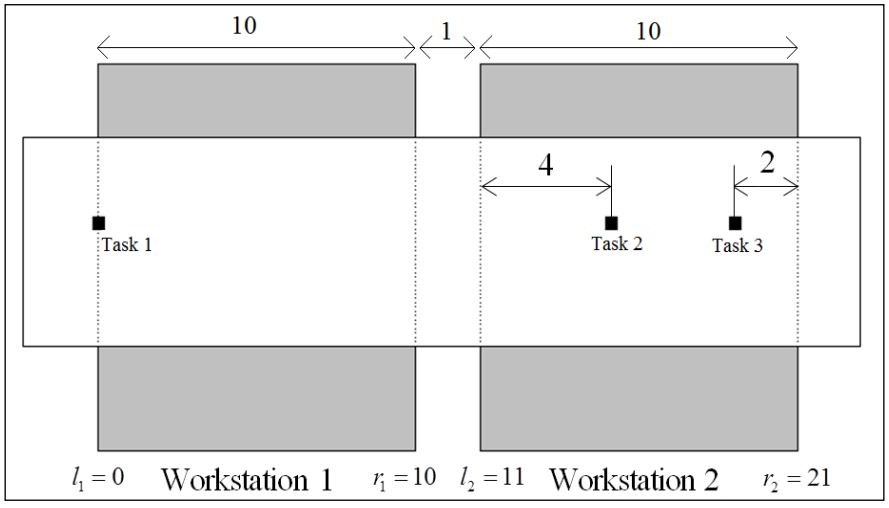

Figure 4. Example illustrating the task accessibility windows

Generalizing the above example, the concept of *accessibility window of a task* emerges, which is defined taking its preceding and succeeding tasks into account, as well as the workstation at which the task must be processed. Equations (1) and (2) formulate the accessibility windows for each task $j \in J$, $\left[lt_j^{PR}, rt_j^{PR} \right]$.

$$lt_j^{PR} = \begin{cases} l_{w_j} & \text{if } PT_j^I = \varnothing \\ \max\left(l_{w_j}, \max_{k \in PT_j^I} \left(lt_k^{PR} + \left(a_k - a_j \right) \right) \right) & \text{otherwise} \end{cases} \qquad j \in J \qquad (1)$$

$$rt_j^{PR} = \begin{cases} r_{w_j} & \text{if } ST_j^I = \varnothing \\ \min\left(r_{w_j}, \min_{k \in ST_j^I} \left(rt_k^{PR} - \left(a_j - a_k \right) \right) \right) & \text{otherwise} \end{cases} \qquad j \in J \qquad (2)$$

The accessibility window of task j $\left[lt_j^{PR}, rt_j^{PR} \right]$ is defined taking into account only the precedence relationships and the accessibility windows of the workstations. Additionally, the movement scheme can be taken into account for better accuracy. Note that the first and last positions in which task j can be processed throughout a cycle (for a given movement scheme) may be greater than lt_j^{PR} and/or smaller than rt_j^{PR}, respectively. Thus, a narrower accessibility window $\left[lt_j, rt_j \right]$ for each task $j \in J$ is defined by Eqs. (3) and (4):

$$lt_j = \begin{cases} \theta^m\left(j, l_{w_j} \right) & \text{if } PT_j^I = \varnothing \\ \theta^m\left(j, \max\left(l_{w_j}, \max_{k \in PT_j} \left(lt_k + \left(a_k - a_j \right) \right) \right) \right) & \text{otherwise} \end{cases} \qquad j \in J \qquad (3)$$

$$rt_j = \begin{cases} \theta^M\left(j, r_{w_j} \right) & \text{if } ST_j^I = \varnothing \\ \theta^M\left(j, \min\left(r_{w_j}, \min_{k \in ST_j} \left(rt_k - \left(a_j - a_k \right) \right) \right) \right) & \text{otherwise} \end{cases} \qquad j \in J \qquad (4)$$

$\theta^m\left(j, \gamma \right)$ ($\theta^M\left(j, \gamma \right)$) is the smallest (greatest) position in which task j will be situated during the cycle, according to the movement scheme, which is greater (smaller) than or equal to position γ. They are defined by Eqs. (5) and (6), respectively, where $\lceil n \rceil$ ($\lfloor n \rfloor$) is the smallest (greatest) integer value that is greater (smaller) than or equal to n, and $d_{j\sigma}$ is the position of

task j during stationary stage σ, corresponding to the first workpiece that is totally or partially visible in the first workstation at the start of each cycle (Eq. 7).

$$\theta^m(j,\gamma) = \min_{\sigma \in S}\left(d_{j\sigma} + b \cdot \left\lceil \left(\gamma - d_{j\sigma}\right)/b \right\rceil\right) \qquad j \in J \qquad (5)$$

$$\theta^M(j,\gamma) = \max_{\sigma \in S}\left(d_{j\sigma} + b \cdot \left\lfloor \left(\gamma - d_{j\sigma}\right)/b \right\rfloor\right) \qquad j \in J \qquad (6)$$

$$d_{j\sigma} = x + \sum_{\tau=1}^{\sigma-1} \delta_\tau - a_j \qquad j \in J\ ;\ \sigma \in S \qquad (7)$$

Values of lt_j and rt_j (also values of lt_j^{PR} and rt_j^{PR}) are defined recursively. The code to calculate them starts with those tasks without precedence tasks (base case), then with the tasks that have 1 level of precedence tasks, followed by the tasks that have 2 levels of precedence tasks, and so on.

3.2. Pre-processing rule 2 - Some precedence relationships do not need to be explicitly imposed

For each $(j,k) \in PR$, let the following cases be discerned according to the relations between the positions of the tasks on the workpiece and the workstations at which the tasks are performed.

C1. $(a_k \geq a_j) \wedge (w_k > w_j)$: The precedence relationship is fulfilled without the need of imposing it explicitly.

C2. $(a_k \geq a_j) \wedge (w_k \leq w_j)$. Three subcases:

C2.1. $a_k - a_j < lt_j - rt_k$: The precedence relationship cannot be satisfied.

C2.2. $lt_j - rt_k \leq a_k - a_j \leq rt_j - lt_k$: The precedence relationship has to be imposed.

C2.3. $a_k - a_j > rt_j - lt_k$: The precedence relationship is fulfilled without the need of imposing it explicitly.

C3. $(a_k < a_j) \wedge (w_k \geq w_j)$. Three subcases:

C3.1. $a_j - a_k < lt_k - rt_j$: The precedence relationship is fulfilled without the need of imposing it explicitly.

C3.2. $lt_k - rt_j \le a_j - a_k \le rt_k - lt_j$: The precedence relationship has to be imposed.

C3.3. $a_j - a_k > rt_k - lt_j$: The precedence relationship cannot be satisfied.

C4. $\left(a_k < a_j\right) \wedge \left(w_k < w_j\right)$: The precedence relationship cannot be satisfied.

Thus we can split set PR into the following disjoint sets PR^1, PR^2 and PR^3:

- $PR^1 = \left\{\left(j,k\right) \in PR : lt_j - rt_k \le a_k - a_j \le rt_j - lt_k\right\}$. The set of precedence relationships that have to be imposed (cases C2.2 and C3.2).
- $PR^2 = \left\{\left(j,k\right) \in PR : a_k - a_j > rt_j - lt_k\right\}$. The set of precedence relationships that are ensured without the need of imposing them explicitly (cases C1, C2.3 and C3.1).
- $PR^3 = \left\{\left(j,k\right) \in PR : a_k - a_j < lt_j - rt_k\right\}$. The set of precedence relationships that cannot be satisfied (cases C2.1, C3.3 and C4).

Note that an instance is feasible if and only if $PR^3 = \varnothing$. Thus, the feasibility of an instance is checked during this pre-processing procedure.

4. Mathematical model

The indices, additional data and variables used in the model are presented in Tables 3, 4 and 5, respectively.

i	Index for the workstations
j, k	Indices for the tasks
σ	Index for the stationary stages
o	Index for the order in which the tasks are performed

Table 3. Indices

Π_j	Set of stationary stages in which task $j \in J$ is within the accessibility window of its workstation, $\left[lt_j, rt_j\right]$: $\Pi_j = \left\{\sigma \in S : d_{j\sigma} + b \cdot \left\lceil \left(lt_j - d_{j\sigma}\right)/b \right\rceil \le rt_j \right\}$
$h_{j\sigma}$	Value needed to ensure the precedence relationships, which is calculated as follows: $h_{j\sigma} = \left\lceil \left(lt_j - d_{j\sigma}\right)/b \right\rceil$. Thus, if $h_{j\sigma} = h_{k\sigma}$ ($j \in J$, $k \in J \setminus \{j\}$, $\sigma \in \Pi_j \cap \Pi_k$) then tasks j and k are accessible to their respective workstations on the same workpiece at stage σ; otherwise, tasks j and k are accessible but on different workpieces
lb^Z	Lower bound of the objective function. It is the maximum value of the following two: 1) optimal value of the problem relaxing the precedence relationships constraints, which is obtained with the MILP model proposed in Calleja *et al.* (2014) (this model is solved very quickly); 2) the lower bound obtained when solving the MILP model described in Section 5.1 (which is a part of the proposed matheuristic procedure, with its solution time limited to 5 seconds)
ub^Z	Upper bound on the objective function, which is a big enough value for Eqs. 14 and 15. In this work we use the value of the solution obtained with the proposed matheuristic (see Section 5)

Table 4. Additional data

$C_\sigma \ge 0$	Time needed for all workstations to complete the tasks assigned to them in stationary stage $\sigma \in S$
$y_{j\sigma} \in \{0,1\}$	1 if task $j \in J$ is processed during stationary stage $\sigma \in \Pi_j$; otherwise 0
$t_{j\sigma} \ge 0$	In the case that task $j \in J$ is processed during stationary stage $\sigma \in \Pi_j$, $t_{j\sigma}$ is the instant in which task j commences to be processed (considering instant 0 as the start of stationary stage σ); otherwise, $t_{j\sigma}$ is meaningless
$q_{j\sigma o} \in \{0,1\}$	1 if task $j \in J$ is the *o-th* task ($o = 1,..., \overline{J}_{w_j}$, where $\overline{J}_i$ is the cardinal of set J_i) to be performed during stationary stage $\sigma \in \Pi_j$; otherwise 0

Table 5. Variables

We propose the following MILP model for the problem stated in Section 2:

$$\left[\text{MIN}\right] \text{Z} = \sum_{\sigma \in S} C_\sigma \tag{8}$$

$$lb^Z \le \sum_{\sigma \in S} C_\sigma \tag{9}$$

$$\sum_{\sigma \in S} C_\sigma \le ub^Z - 1 \tag{10}$$

$$\sum_{\sigma \in \Pi_j} y_{j\sigma} = 1 \qquad j \in J \tag{11}$$

$$C_\sigma \ge t_{j\sigma} + p_j \cdot y_{j\sigma} \qquad j \in J\ ;\ \sigma \in \Pi_j \tag{12}$$

$$\left(ns \cdot \sum_{\sigma \in \Pi_j} h_{j\sigma} \cdot y_{j\sigma} \right) + \sum_{\sigma \in \Pi_j} \sigma \cdot y_{j\sigma} \leq \left(ns \cdot \sum_{\sigma \in \Pi_k} h_{k\sigma} \cdot y_{k\sigma} \right) + \sum_{\sigma \in \Pi_k} \sigma \cdot y_{k\sigma} \qquad (j,k) \in PR^1 \qquad (13)$$

$$t_{k\sigma} \geq t_{j\sigma} + p_j - ub^Z \cdot \left(2 - y_{j\sigma} - y_{k\sigma}\right) \qquad (j,k) \in PR^1;\ \sigma \in \Pi_j \cap \Pi_k : h_{j\sigma} = h_{k\sigma} \qquad (14)$$

$$t_{k\sigma} \geq t_{j\sigma} + p_j - ub^Z \cdot \left(2 - q_{j,\sigma,o-1} - q_{k\sigma o}\right)$$
$$i \in I;\ j \in J;\ k \in J_i \setminus \{j\};\ \sigma \in \Pi_j \cap \Pi_k;\ o = 2,...,\overline{J}_i \qquad (15)$$

$$\sum_{o=1}^{\overline{J}_{w_j}} q_{j\sigma o} = y_{j\sigma} \qquad j \in J;\ \sigma \in \Pi_j \qquad (16)$$

$$\sum_{j \in J_i : \sigma \in \Pi_j} q_{j\sigma o} \leq 1 \qquad i \in I;\ \sigma \in S;\ o = 1,...,\overline{J}_i \qquad (17)$$

$$\sum_{j \in J_i : \sigma \in \Pi_j} q_{j\sigma o} \leq \sum_{j \in J_i : \sigma \in \Pi_j} q_{j,\sigma,o-1} \qquad i \in I;\ \sigma \in S;\ o = 2,...,\overline{J}_i \qquad (18)$$

The objective function to minimise (8) is the sum, for all stationary stages, of the maximum completion times of the tasks performed in each stage. Constraint (9) means that the solution value is equal to or greater than the lower bound value. Constraint (10) leads to the returned solution value being strictly better than the value of the solution found with the matheuristic; therefore, the matheuristic solution is optimal if the model has no feasible solutions. Constraints (11) ensure that each task is performed in exactly one stationary stage. Constraints (12) guarantee that the time of any stationary stage is not less than the completion time of any task performed during that stationary stage. Constraints (13) enforce the fulfilment of the precedence relationships of the pairs of tasks in PR^1 as follows. Note that $\sum_{\sigma \in \Pi_j} \sigma \cdot y_{j\sigma} \leq ns$ by definition. Thus, at each cycle, predecessor task j is performed on a workpiece placed on the left of the workpiece at which successor task k is performed (i.e, $\sum_{\sigma \in \Pi_j} h_{j\sigma} \cdot y_{j\sigma} < \sum_{\sigma \in \Pi_k} h_{k\sigma} \cdot y_{k\sigma}$) or, if they are performed on the same workpiece (i.e., if $\sum_{\sigma \in \Pi_j} h_{j\sigma} \cdot y_{j\sigma} = \sum_{\sigma \in \Pi_k} h_{k\sigma} \cdot y_{k\sigma}$), then task j is performed during a stationary stage no later than the stationary stage in which task k is performed (i.e., $\sum_{\sigma \in \Pi_j} \sigma \cdot y_{j\sigma} \leq \sum_{\sigma \in \Pi_k} \sigma \cdot y_{k\sigma}$). Thus, given $(j,k) \in PR^1$, at each cycle either task j is executed on a workpiece placed on the left of the workpiece on which k is performed (i.e, $\sum_{\sigma \in \Pi_j} h_{j\sigma} \cdot y_{j\sigma} < \sum_{\sigma \in \Pi_k} h_{k\sigma} \cdot y_{k\sigma}$) or they are performed on the same workpiece (i.e., if $\sum_{\sigma \in \Pi_j} h_{j\sigma} \cdot y_{j\sigma} = \sum_{\sigma \in \Pi_k} h_{k\sigma} \cdot y_{k\sigma}$) with j being performed at a stationary stage no later than the stationary stage at which k is performed (i.e., $\sum_{\sigma \in \Pi_j} \sigma \cdot y_{j\sigma} \leq \sum_{\sigma \in \Pi_k} \sigma \cdot y_{k\sigma}$). Constraints (14)

ensure that each task k commences to be processed later than each precedent task j has been processed if both tasks are performed during the same stationary stage σ (i.e., if $y_{k\sigma} = y_{j\sigma} = 1$); otherwise, note that ub^Z is a big enough value to satisfy the constraints. Constraints (15) ensure that the start time of the tasks of the same workstation and stationary stage is consequent to the tasks order defined with variables $q_{j\sigma o}$. Thus, if task k is performed immediately after task j at the same workstation and stationary stage (i.e., $q_{j,\sigma,o-1} = q_{k,\sigma,o} = 1$), then (15) guarantee that the start time of task k (i.e., $t_{k\sigma}$) is not less than the finish time of task j (i.e., $t_{j\sigma} + p_j$). Constraints (16-18) ensure an order (sequence) of tasks to be performed at each workstation and stationary stage. Specifically, (16) enforce that each task is assigned to one and only one ordinal position in the sequence of its workstation tasks and the stationary stage in which it is processed. (17) guarantee that no more than one task is assigned to an ordinal position. Finally, (18) impose that the sequence of tasks is filled without empty positions between the first and the last occupied position. It is worth pointing out that constraints (12)-(18) are new in the literature, since this is the first time that precedence relationships are taken into account in the AWALBP.

We designed other alternatives for modelling the precedence constraints (Eq, 13), which are based on two ways of modelling the precedence relationships for SALBP. One alternative is based on Bowman (1960) as follows:

$$\left(\sum_{\tau\in\Pi_j} h_{j\tau}\cdot y_{j\tau}\right) + y_{k\sigma} \le \left(\sum_{\tau\in\Pi_k} h_{k\tau}\cdot y_{k\tau}\right) + \left(\sum_{\tau\in\Pi_j:\tau\le\sigma} y_{j\tau}\right) \qquad (j,k)\in PR^1,\ \ \sigma\in\Pi_k$$

Another alternative is based on Ritt and Costa (2015) as follows:

$$\left(\sum_{\tau\in\Pi_j} h_{j\tau}\cdot y_{j\tau}\right) + \left(\sum_{\tau\in\Pi_k:\tau\le\sigma} y_{k\tau}\right) \le \left(\sum_{\tau\in\Pi_k} h_{k\tau}\cdot y_{k\tau}\right) + \left(\sum_{\tau\in\Pi_j:\tau\le\sigma} y_{j\tau}\right) \qquad (j,k)\in PR^1,\ \ \sigma\in\Pi_k$$

Slightly worse results were obtained with the alternative constraints in an initial computational experiment (specifically, on average, 0.61% and 0.58% worse cycle times). Thus, the computational experiment reports the results obtained using Eq. 13.

Since the processing times are integer, the objective function value is also integer. This allows the absolute gap to be set at 1-ε (where $\varepsilon = 10^{-6}$) in CPLEX when solving the model and thus the computational time may be reduced.

5. Matheuristic procedure

As we explained, the matheuristic procedure is applied before the MILP model proposed in Section 4. The flowchart of the overall solution scheme is illustrated in Figures 5 (pre-process), 6 (matheuristic) and 7 (MILP model).

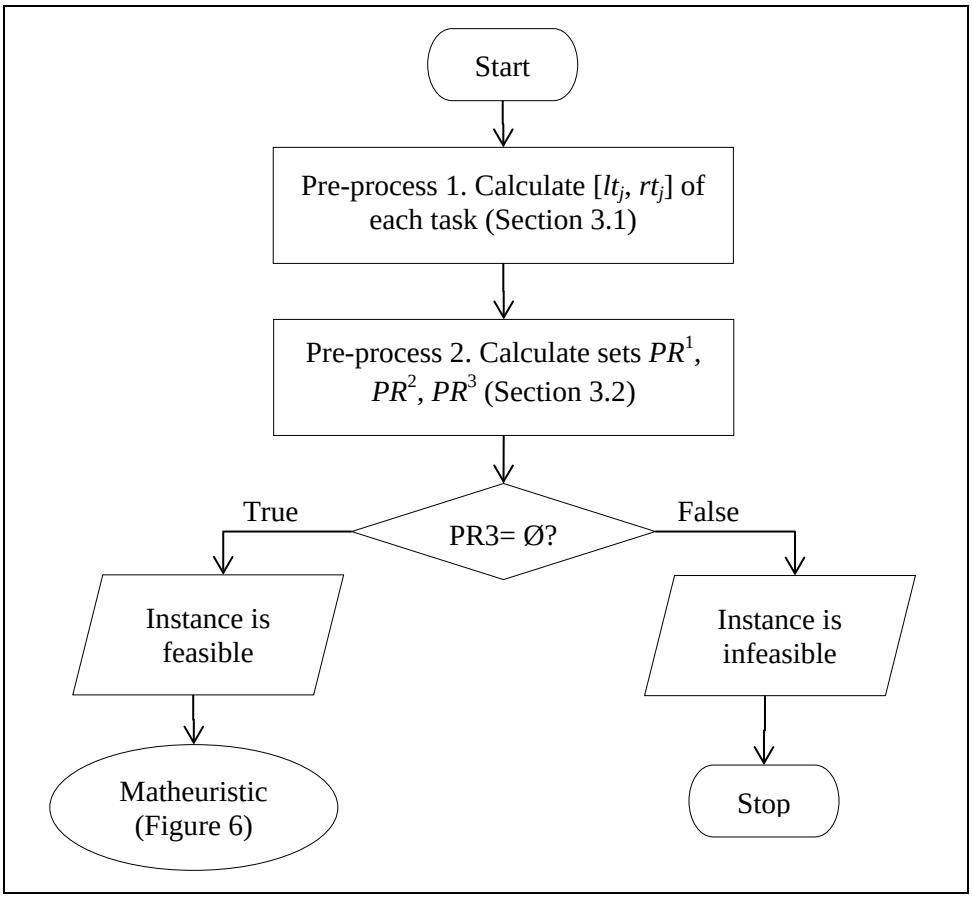

Figure 5. Flowchart of the pre-process procedure

The pre-processing procedures not only reduce the size of the instance but also detect whether the instance is infeasible. In this case, the solution process stops; otherwise, the matheuristic is applied and a solution is obtained.

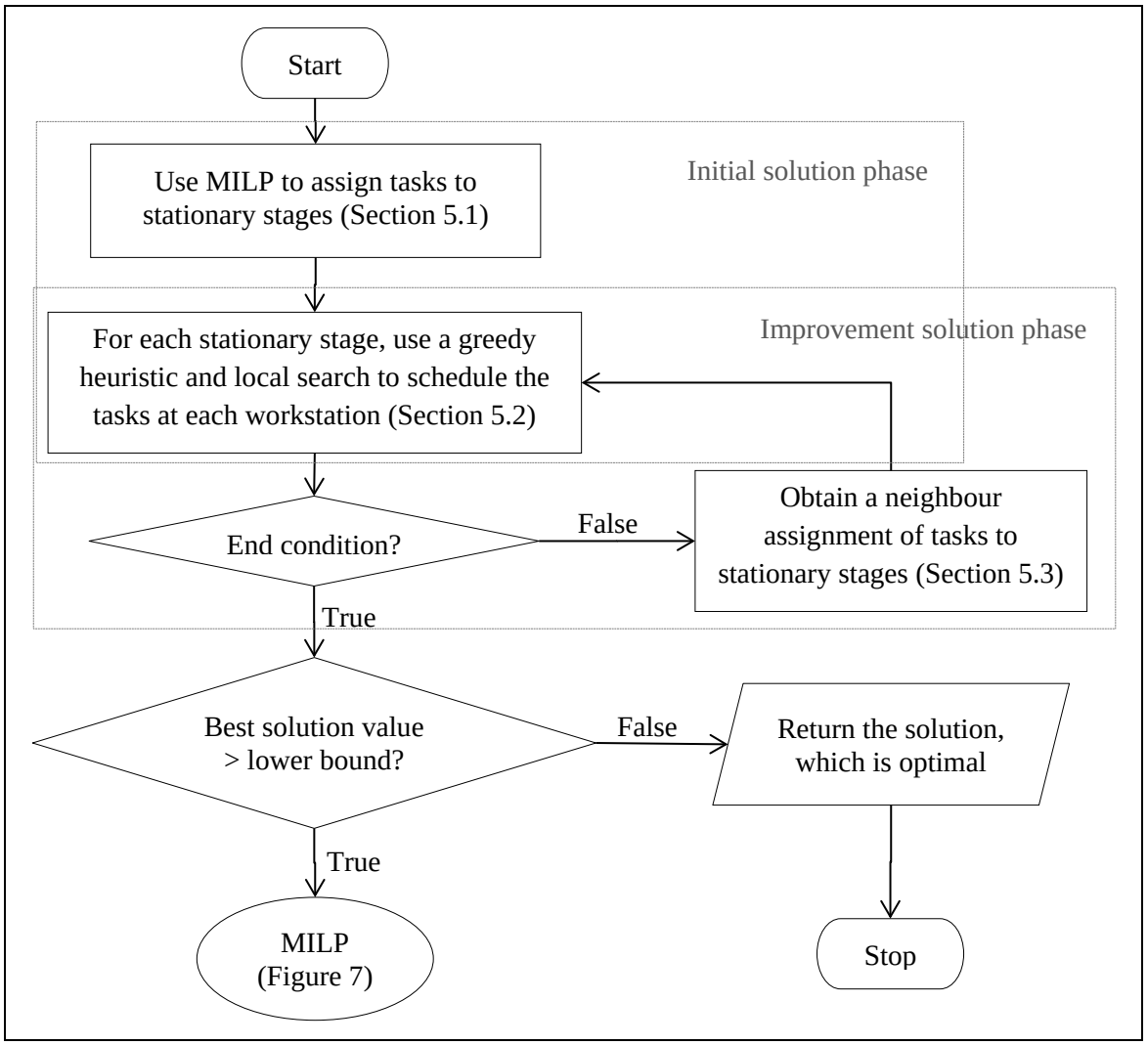

Figure 6. Flowchart of the matheuristic procedure

The matheuristic (which will be explained in detail later) starts calculating, in two steps, an initial solution (initial solution phase). In the first step, the assignment of tasks to stationary stages is decided with a MILP model. In the second step, the scheduling of the tasks to be performed at each pair (workstation, stationary stage) is decided with a greedy heuristic and a local search procedure. Then, iteratively, a neighbour solution is calculated (improvement solution phase). Specifically, a neighbour assignment of tasks is obtained and the new scheduling of tasks is calculated again with the greedy heuristic and the local search procedure.

If the value of the best solution obtained with the matheuristic is equal to the lower bound, then its optimality is demonstrated and the solution procedure stops. Otherwise, a MILP model is solved to calculate a better solution. There are four possible outcomes when trying to solve the model (Figure 7): 1) the infeasibility of the model is demonstrated and therefore the matheuristic solution is optimal, 2) the infeasibility of the model is not demonstrated but a better

solution is not found, 3) a better solution is found but its optimality is not demonstrated, and 4) a (demonstrated) optimal solution is found.

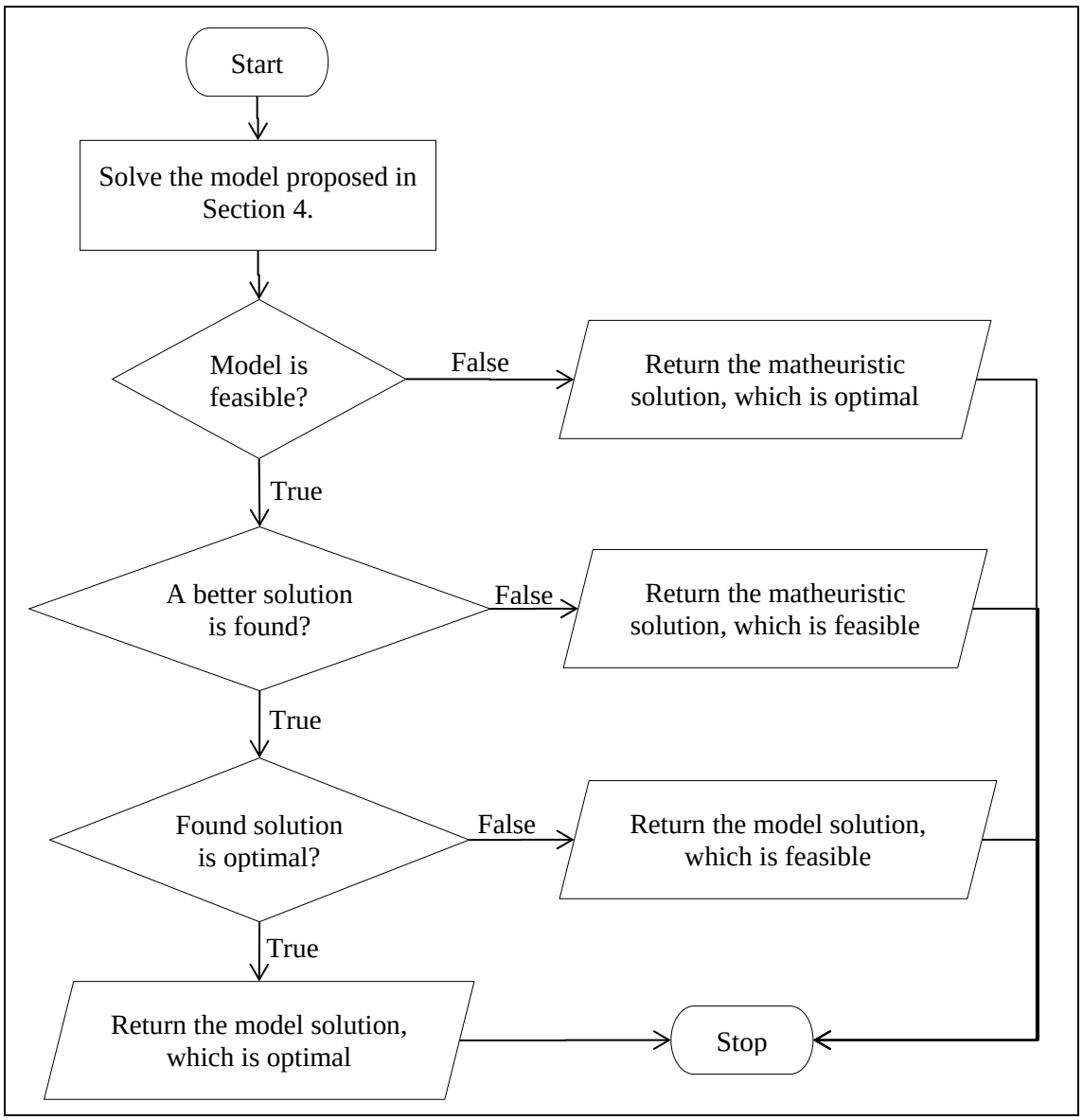

Figure 7. Flowchart of the possible outcomes of the MILP model

In designing the matheuristic procedure, the quality of the obtained solutions and the computing time have been taken into account. The efficiency of this procedure is important because it may be used in future research into AWALBP-L1-M and upper levels of AWALBP.

The main framework of the proposed matheuristic is based on the simulated annealing metaheuristic (SA). SA has been applied successfully to solve different combinatorial optimisation problems (Nikolaev and Jacobson, 2010). For instance, in García-Villoria *et al.* (2012), 13 procedures are compared to solve the response time variability problem; the best one is based on SA. This metaheuristic is also used with successful results to solve assembly line

balancing problems (Battaïa and Dolgui, 2013). In particular, SA has been used to solve AWALBP (Calleja *et al*., 2016; García-Villoria *et al*., 2015).

As shown in Figure 6, SA starts from an initial solution. Then, iteratively, a neighbour solution selected at random from the neighbourhood of the current solution is considered. Moves to non-worse neighbours are always accepted. On the other hand, in order to escape from local optima, moves to worse neighbours are accepted with a certain probability that depends on the quality of the neighbour and on a parameter called temperature (t), whose value is dynamically decreased.

In our procedure, the neighbourhood of a solution is not defined in the space of complete solutions but in the space of the assignment of tasks to stationary stages (Section 5.3). To obtain an initial assignment of tasks to stationary stages, *TtoST*, a MILP model is used (Section 5.1). And for a given *TtoST* a heuristic is applied to set the scheduling of the tasks, *SchT* (Section 5.2). Thus, the pair (*TtoST*, *SchT*) defines a complete solution. The pseudocode of the matheuristic procedure is detailed in Figure 8, where $Z(TtoST, SchT)$ returns the Z value (Eq. 8) of the solution (*TtoST*, *SchT*).

The matheuristic has 4 parameters (t_0, t_f, itt and α, see Figure 8). To set the values of these parameters, we have used CALIBRA (Adenso-Díaz and Laguna, 2006). CALIBRA is an automatic tool based on Taguchi's fractional factorial experimental design and local search for fine-tuning parameters of algorithms. The training set for the fine-tuning has 60 representative instances. The way in which they are generated is explained in Section 6.1. The values obtained with CALIBRA are the following: $t_0 = 260$, $t_f = 0.01$, $itt = 130$ and $\alpha = 0.99$. The obtained parameter values seem suitable for the convergence of the matheuristic. For example, Figures 9 and 10 show the values of the best solution found every 1,000 iterations of the matheuristic when solving a test instance that requires little computing time (15.04 s, Figure 9) and an instance that requires much more computing time (798.16 s, Figure 10).

1. $TtoST^*$ = Calculate an assignment of tasks to stationary stages (Sect. 5.1)
2. $SchT^*$ = Given $TtoST^*$, calculate the scheduling of tasks (Sect. 5.2)
3. Set the values of parameters:
   t_0 (initial temperature)
   t_f (final temperature)
   itt (number of iterations in which the temperature remains equal)
   α (cooling factor): $0 < \alpha < 1$
4. $(cTtoST, cSchT) = (TtoST^*, SchT^*)$
5. **For** $(t = t_0;\ (t \geq t_f) \wedge (Z(TtoST^*, SchT^*) > lb^Z);\ t = \alpha \cdot t)$ **do**:
6. **For** $(i = 1;\ (i \leq itt) \wedge (Z(TtoST^*, SchT^*) > lb^Z);\ i = i + 1)$ **do**:
7. $nTtoST$ = Select a neighbour of $cTtoST$ (Sect. 5.3)
8. $nSchT$ = Given $nTtoST$, calculate the scheduling of tasks (Sect 5.2)
9. $\Delta = Z(nTtoST, nSchT) - Z(cTtoST, cSchT)$
10. **If** $(\Delta \leq 0)$ **then** $(cTtoST, cSchT) = (nTtoST, nSchT)$
11. **otherwise** $(cTtoST, cSchT) = (nTtoST, nSchT)$ with probability $\exp(-\Delta/t)$
12. **End if**
13. **If** $Z(cTtoST, cSchT) < Z(TtoST^*, SchT^*)$ **then** $(TtoST^*, SchT^*) = (cTtoST, cSchT)$ **End if**
14. **End for**
15. **End for**
16. **Return** $(TtoST^*, SchT^*)$

Figure 8. Framework of the proposed matheuristic

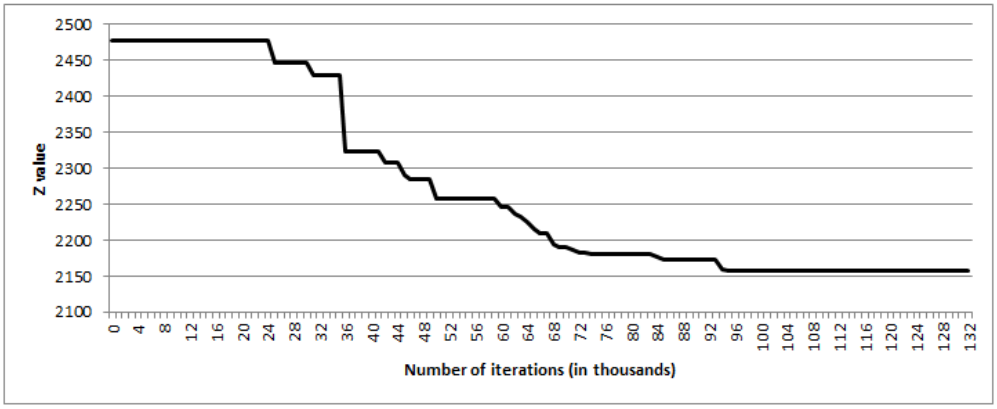

Figure 9. Convergence of a quick to solve instance

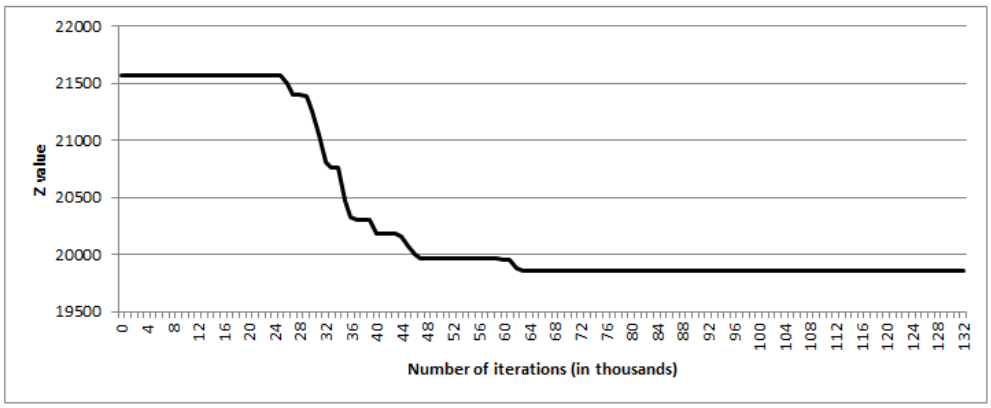

Figure 10. Convergence of a slow to solve instance

In any case, although the training set contains instances of different characteristics, it would be advisable to tune the parameters when dealing with other scenarios for a potentially better performance.

Section 5.1. Initial assignment of tasks to stationary stages

To calculate the initial assignment of tasks to stationary stages we apply a MILP model with variables C_σ and $y_{j\sigma}$ (recall Section 4), the objective function formulated by Eq. (8), the constraints corresponding to Eqs. (9), (11), (13), and the following Eq. (12'):

$$C_\sigma \geq \sum_{j \in J_i : \sigma \in \Pi_j} p_j \cdot y_{j\sigma} \qquad i \in I\,;\ \sigma \in S \qquad (12')$$

This model (contrary to what happens in the case of the model proposed in Section 4) does not take into account the potential idle times (due to precedence relationships) between the processing of the tasks. Note that variables $t_{j\sigma}$ and $q_{j\sigma o}$ do not figure in this model, which, therefore, does not include Eqs. (14) to (18). We use this simplified model instead of the original one because no (feasible) solutions can be obtained with the latter for instances with more than 200 tasks.

Although the model proposed in this subsection is simpler than the model shown in Section 4, it is still very hard to solve optimally for instances with a non-small number of precedence relationships (see Section 6.2). On the other hand, in order to have an efficient overall matheuristic procedure, the initial assignment should be calculated quickly. Thus, the solution of the model is limited by 5 computing seconds (with an absolute gap equal to 1-ε), which is

enough time to obtain a feasible solution even for the biggest instances. This is justified by the following two reasons:

- This initial assignment of tasks to stationary stages is a heuristic step of the matheuristic, so obtaining the optimal solution of the model does not seem to be critical.

- Moreover, a previous computational experiment shows that the quality of the best solution found by the matheuristic is quite robust with respect to the initial assignment.

Section 5.2. Scheduling of tasks

For a given assignment of tasks to stationary stages, we have to schedule the tasks for each stationary stage. Note that the completion time of each stationary stage is independent of the completion times of the other stationary stages. Thus, the problem of deciding the scheduling of tasks in each stationary stage can be solved independently.

The scheduling problem to solve for each stationary stage is equivalent to minimise the duration of a project with limited resources, where tasks represent activities, workstations represent resources and the time to process all tasks (in that stationary stage) is the project horizon. To solve it, the well-known heuristic based on the disjunctive graph model representation (see Roy and Sussmann, 1964) is used and then a local search (LS) procedure is applied to the obtained solution.

The heuristic schedules the tasks as follows. Iteratively, for each pair a and b of activities (tasks) that share the same resource (workstation), the values ς_{ab} and ς_{ba} are calculated, where $\varsigma_{jk} = t_j + p_j + T - T_k$, t_j and T_j are the minimum and maximum time instant, respectively, in which the activity can start to be performed, and T is the minimum project horizon so that all activities can be performed. At each iteration, let a^* and b^* be the pair of activities with the biggest ς_{jk} value so the addition of the arc $\left(b^*, a^*\right)$ would not create a cycle in the precedence graph; then arc $\left(b^*, a^*\right)$ is added. The heuristic stops when, for each resource i, the precedence relationships between activities that share resource i define a sequence (that is, an order of the activities).

The proposed LS procedure uses a non-exhaustive exploration strategy (i.e., at each iteration the first neighbour that improves the current solution is selected) and uses a neighbourhood structure based on the *critical end transpose* (CET) neighbourhood (Nowicki and Smutnicki,

1996). The CET neighbourhood was originally proposed as part of a tabu search procedure for solving the job shop problem. Here we adapt the CET neighbourhood as follows. The tasks of a critical (longest) path can be decomposed into blocks. A block is defined as a sequence of activities that share a resource connected by the longest chain. Let SA be the set of arcs that connect two blocks of a critical path. For each arc $(a,b) \in SA$, a neighbour is generated by exchanging the positions in the sequence of activity a and of any immediately previous activity that shares its resource (obviously, only if the original precedence relationships are fulfilled). For instance, let $(\beta,\kappa) \in SA$ and $(\alpha,\beta) \notin SA$, $w_\alpha = w_\beta \neq w_\kappa$ (i.e., activities α and β share the same resource, but activity κ does not), and $(\alpha,\beta) \notin PR$ and $(\beta,\alpha) \notin PR$ (i.e., there are no precedence relationships between tasks α and β). Then suppose that activity α is immediately scheduled before activity β. Thus, a neighbour is obtained by exchanging the positions of activities α and β.

Another way of scheduling the tasks is by solving a MILP model. Specifically, the model proposed in Section 4 could be used, in which the values of variables $y_{j\sigma}$ are set according to the given assignment of tasks to stationary stages. However, preliminary experiments showed that this model is hard to solve and too much computing time is required even for finding feasible solutions.

Section 5.3. Neighbourhood of an assignment of tasks to stationary stages

Given a current assignment of tasks to stationary stages ($TtoST$), let $TtoST_j$ be the stationary stage at which task j is assigned. A neighbour of $TtoST$ is obtained with the following movement: the assignment of one task $j \in J : |\Pi_j| \geq 2$ is changed to one stationary stage $\sigma \in \Pi_j \setminus \{TsoST_j\}$ so that the precedence relationships are fulfilled. Thus, an upper bound of the size of the neighbourhood is $nt \cdot ns$. Note that because task j will be accessible to its workstation and the precedence relationships can be fulfilled, all obtained neighbours are feasible.

6. Computational experiment

All MILP models were solved with IBM ILOG CPLEX 12.6. The matheuristic was implemented in Java SE 1.6.21. The experiments were run on a PC 3.33 GHz Pentium Intel Core i5 with 4 GB RAM.

Section 6.1 describes the training and test instances solved in the computational experiment. Section 6.2 analyses the obtained results.

6.1. Test and training instances

Since the AWALBP with precedence relationships between tasks has not been dealt with in the literature, there is no benchmark set of instances. Calleja *et al*. (2013, 2014, 2016) used 1,200 test instances, available at https://www.ioc.upc.edu/EOLI/research/, to test their procedures for AWALBP-L2-1 without precedences between tasks. In this work, test and training instances are generated for AWALBP-L1-1 with precedence relationships based on those 1,200 AWALBP-L2-1 instances, in which the following data are added: precedence relationships and a movement scheme.

A part of the data that define the 1,200 AWALBP-L2-1 instances is generated as follows, (where $U[\cdot]$ refers to the discrete uniform distribution): $b_0 = U[11,40]$, $b \in \{b_0, b_0+1\}$, $nw = U[5,40]$, $l_i = 11\cdot(i-1)$ and $r_i = l_i + 10$ ($i \in I$), $nt = U[50,1000]$, $w_j = U[1,nw]$, $a_j = U[0,b_0]$ and $p_j = U[100,150]$ ($j \in J$). For more details, see Calleja *et al*. (2013).

From each of the existing 1,200 instances, a new set of instances is generated with precedence characteristics that may appear in the assembly of PCBs. The values of b_0, b, nw, l_i, r_i, nt, w_j, a_j and p_j in each new instance are equal to those of its original instance. The precedence relationships are added with the following characteristics. For each task $j \in J$, the total number of immediate successors of task j is not greater than 4 (i.e., $|ST_j^I| \le 4$), there is at most one level of successors (i.e., if $ST_j^I \neq \varnothing$ then $\forall k \in ST_j^I : ST_k^I = \varnothing$) and each task j has at most one (immediate) predecessor (i.e., $|PT_j^I| \le 1$). Moreover, there can be precedence relationships only between very close tasks; specifically, if $(j,k) \in PR$ then $|a_j - a_k| \le 1$. The total number of precedence relationships is $0.02 \cdot nt$; therefore, the order strength value is $0.04/(nt-1)$. Order strength (OS) is a well-known metric of the density of the precedence graph that is defined as $OS = 2 \cdot \sum_{j \in J} |PT_j^T| \Big/ nt \cdot (nt-1)$. Finally, the movement scheme is selected from 10 movement schemes generated at random. The selection criterion is the lowest number of forward steps; in case of a tie, it is resolved in favour of the less *irregular* solution (the irregularity is quantified as $\sum_{\sigma \in S} \left(\delta_\sigma - \frac{b}{ns} \right)^2$). A low number of stationary stages tends to favour evenly balanced

workloads in each stationary stage (moreover, it reduces the acceleration and deceleration times corresponding to the forward steps of the line). Each random movement scheme is generated as follows. The initial shift x is selected at random (with equal probability) between 0 and $b-1$. The number of forward steps, ns, and their lengths, δ_σ, are set as follows. Iteratively, until $\sum_{\sigma \in S} \delta_\sigma = b$, the length of the current forward step σ is selected at random (with equal probability) between 1 and the minimum of these values: $b - \sum_{l=1}^{\sigma-1} \delta_l$ and the maximum length of δ_σ so each task j can be accessible in its workstation. Let this new set of 1,200 instances be called *PCB test set*.

Additionally, another set of test instances with bigger OS is generated. From each of the original AWALBP-L2-1 instances, 3 new test instances are generated with different OS values: $OS \approx 0.1$, $OS \approx 0.4$ and $OS \approx 0.8$. The movement scheme is generated in the same way as for the PCB test set instances. The algorithm for adding precedence relationships is shown in Figure 11, where $OS(PR)$ is the OS value corresponding to precedences PR. The algorithm is based on randomly adding an immediate precedence relationship so that it does not generate cycles in the precedence graph and it allows at least one feasible movement scheme (i.e., $PR^3 = \varnothing$). Let this set of 3,600 instances be called *general test set*.

1. Let $\widehat{OS}$ be the desired OS ($\widehat{OS} \in \{0.1, 0.4, 0.8\}$)
2. $PR = \varnothing$
3. $C = J \times (J \setminus \{j\})$: Set of potential pairs of precedence relationships
4. **While** $(OS(PR) < \widehat{OS}) \wedge (C \neq \varnothing)$ **do**:
5. $(\hat{j}, \hat{k})$ = Select at random an element from C; $C = C \setminus \{(\hat{j}, \hat{k})\}$
6. $PR = PR \cup \{(\hat{j}, \hat{k})\}$; Calculate lt_j^{PR} and rt_j^{PR} for all $j \in J$
7. **If** ($PR^3 \neq \varnothing$) $\vee$ (precedence graph defined by PR is cyclic)
   then: $PR = PR \setminus \{(\hat{j}, \hat{k})\}$
   else: Remove redundant pairs of precedence relationships from PR
   End if
8. **End while**
9. **Return** PR

Figure 11. Algorithm for generating precedence relationships

Finally, a training set of 60 instances is used to fine-tune the matheuristic procedure. This set is generated in the same way as the instances of the general test set.

All sets of instances are available at https://www.ioc.upc.edu/EOLI/research/.

6.2. Computational results

To solve each test instance, the maximum overall computing time was limited to 1 hour. The following solution scheme is applied. First, the instance is pre-processed (Section 3) and then a solution is calculated with the proposed matheuristic (Section 5). If the Z value of the matheuristic solution is equal to lb^Z, then its optimality is demonstrated. Otherwise, the proposed MILP model (Section 4) is solved within the remaining computing time. Section 6.2.1 discusses the solutions obtained without the MILP model. Section 6.2.2 discusses the final solutions provided by the overall solution scheme (i.e., after solving the MILP model).

6.2.1. Results of the matheuristic procedure

Table 6 shows the average Z value of the obtained solutions (*Z*), the average gap of the solutions with respect to lb^Z (*Gap*), the percentage of times that the gap is equal to 0 (*Gap=0*) (i.e., percentage of proven optimal solutions) and the average computing time (*T*), in seconds.

	Z	***Gap***	***Gap=0***	***T (s)***
PCB test set	5436.03	0.03%	99.50%	0.84
General test set	7614.08	19.53%	3.64%	88.81

Table 6. Results obtained by the proposed matheuristic

Our matheuristic procedure performs very well when solving the PCB instances. Except for 6 instances, proven optimal solutions are, on average, found in less than 1 computing second. On the other hand, general instances are harder to solve. The average gap for the general test instances is 19.53%, and 3.64% proven optimal solutions are obtained. One reason for this difference between average gaps could be that the lb^Z values of the general instances are less accurate. Note that lb^Z is calculated assuming that there are no idle times between tasks. Therefore, as PCB instances have fewer precedence relationships than the general instances (and, therefore, less total idle time in the optimal solutions), the lb^Z values of the PCB instances may be expected to be more accurate.

To ascertain if the number of precedence relationships influences the gap, Table 7 shows the results of the general instances split by their order strength (OS). Additionally, the standard deviations of the Z values, gaps and computing times are shown in parenthesis. Effectively, the greater the OS, the greater the gap and fewer proven optimal solutions are obtained. Moreover, Table 7 shows, as could be expected, that the average *Z* is lower when the instance is less

constrained (i.e., it has fewer precedence relationships). The computing time is quite similar regardless of the OS.

	Z	*Gap*	*Gap=0*	T (s)
$OS \approx 0.1$	6070.73 (4427.47)	10.21% (11.07)	8.58%	88.63 (158.25)
$OS \approx 0.4$	6902.17 (4350.01)	19.89% (13.83)	2.33%	81.51 (124.41)
$OS \approx 0.8$	9868.62 (5640.74)	28.50% (11.01)	0.00%	96.28 (265.91)

Table 7. Results of the general instances according to their order strength

Next, the performance of the matheuristic according to the other characteristics of the instances is analysed. The focus is on the general instances, since the matheuristic performs very well for all types of PCB instances. Tables 8, 9 and 10 show, respectively, the results of the test instances split by the length of the workpieces (b_0), the number of workstations (nw) and the number of tasks (nt). For each characteristic, the number of instances in each division is the same.

b_0	Z	*Gap*	*Gap=0*	T (s)
$11 \le b_0 \le 15$	6315.30 (4826.08)	11.15% (12.12)	10.50%	210.65 (422.41)
$16 \le b_0 \le 20$	6980.34 (4984.10)	14.99% (13.04)	5.33%	105.24 (121.44)
$21 \le b_0 \le 25$	7428.31 (4779.08)	18.05% (13.14)	2.83%	75.48 (70.02)
$26 \le b_0 \le 30$	7888.56 (5137.84)	21.58% (13.61)	2.00%	56.90 (49.43)
$31 \le b_0 \le 35$	8280.34 (5026.37)	24.46% (13.49)	1.00%	45.45 (30.07)
$36 \le b_0 \le 40$	8791.66 (5486.31)	26.96% (13.13)	0.17%	39.11 (27.21)

Table 8. Results of the general test instances according to the length of the workpieces

Instances with larger workpiece lengths have greater average gaps and fewer proven optimal solutions. On the other hand, the narrower the workpiece, the greater the computing time spent by the matheuristic, as well as the increase in its deviation.

nw	*Z*	*Gap*	*Gap=0*	*T (s)*
$5 \le nw \le 10$	12139.54 (6579.84)	10.74% (11.34)	10.00%	154.97 (358.87)
$11 \le nw \le 20$	7653.37 (3876.50)	17.90% (13.46)	2.33%	76.49 (76.49)
$21 \le nw \le 30$	5798.28 (2924.63)	23.35% (13.64)	1.67%	64.04 (64.04)
$31 \le nw \le 40$	4865.15 (2491.88)	26.13% (13.02)	0.56%	59.73 (59.73)

Table 9. Results of the general test instances according to the number of workstations

Instances with larger number of workstations have greater average gaps and fewer proven optimal solutions. With a small number of workstations, the computing time and its deviation are clearly greater than those of instances with a higher number of workstations.

nt	*Z*	*Gap*	*Gap=0*	*T (s)*
$50 \le nt \le 200$	3093.24 (1465.99)	29.06% (13.39)	0.97%	17.04 (8.85)
$201 \le nt \le 400$	5613.88 (2719.42)	22.59% (13.34)	2.22%	36.71 (20.37)
$401 \le nt \le 600$	7687.11 (3605.11)	18.49% (12.95)	2.50%	66.44 (40.73)
$601 \le nt \le 800$	9895.35 (5102.43)	15.36% (12.85)	4.31%	120.09 (142.49)
$801 \le nt \le 1000$	11780.94 (5912.11)	12.15% (11.91)	8.19%	203.76 (375.00)

Table 10. Results of the general test instances according to their number of tasks

Regarding the number of tasks, it may seem surprising that the gap (together with its deviation and the number of proven optimal solutions) improves when the number of tasks increases. Several solutions were analysed to explain the phenomenon, leading to the following observations. Instances with few tasks have greater idle times in the workstations (due to the precedence relationships). In contrast, in instances with a great number of tasks the idle times are smaller because there are more possible tasks to fill potential idle times. Thus, the lb^Z values may be more accurate with a larger number of tasks (recall that the calculation of lb^Z assumes that there are no idle times). On the other hand, the computing time and its deviation increase with the number of tasks.

6.2.2. Results of the MILP model

The MILP model is applied to the instances in which the solutions found with the matheuristic have not been proven to be optimal (see Table 6): 0.50% (6) of the PCB (6) and 96.36% (3469),

of the general instances, respectively. CPLEX can return one of the following four responses (see Figure 7): 1) the model is proven to be infeasible and, thus, the optimality of the matheuristic solution is demonstrated; 2) a better solution with demonstrated optimality is found; 3) a better solution whose optimality is not demonstrated is found; and 4) no better solution is found. However, when the aforementioned 6 PCB instances (which have more than 800 tasks) or general instances with more than 200 tasks are solved, in most cases CPLEX does not demonstrate that the model is infeasible, does not find any better solution, or aborts due to memory problems. Thus, we will focus on the general instances with no more than 200 tasks.

According to Table 10, the model was solved for 99.03% (713) of general instances with no more than 200 tasks. Table 11 shows the number of times that CPLEX demonstrates that the model is infeasible (*#Inf*), CPLEX finds a demonstrated optimal solution (*#Opt*), CPLEX finds a feasible (non-demonstrated optimal) solution (*#Feas*) and CPLEX does not find a better solution (*#NBS*). The average gap with respect to the lower bound calculated by CPLEX (which is equal to or better than lb^Z) is given in parenthesis.

#Inf	***#Opt***	***#Feas***	***#NBS***
0	450	66 (4.87%)	197 (18.75%)

Table 11. Results obtained with the MILP model for 713 general instances with no more than 200 tasks

After the solution of the model, 450 new optimal solutions of the general instances are found. However, even for the instances with the lowest number of tasks whose optimal solution was not found with the matheuristic, there are 263 (36.89%) instances that cannot be solved optimally with MILP.

Finally, since several better lower bounds are calculated, the best lower bounds are compared with the objective function values obtained with the matheuristic for the 720 general instances with no more than 200 tasks. Recall that the average gap with respect to lb^Z is 29.06% (see Table 10). On the other hand, the average gap with respect to the improved lower bounds is 5.58% and, thus, the matheuristic procedure performs well for this type of instances. Therefore, this seems to yield more evidence that lb^Z in general may not be very accurate and the matheuristic may perform well for all types of instances.

7. Conclusions

This paper deals with the case of the AWALBP-L1-1 with a single-model line in which there are precedence relationships between tasks. Consequently, idle times between tasks may appear

due to the precedences. Thus, the scheduling of tasks in each workstation influences the production rate of the assembly line together with the assignment of tasks to stationary stages.

Calleja *et al.* (2014) proposed a MILP to solve an AWALBP-L1-1 case without precedence relationships and optimal solutions were found very quickly. On the other hand, the presence of precedence relationships complicates the formulation and solution of the problem and new procedures have been designed for solving this case. Specifically, two pre-processing procedures, a MILP model and a matheuristic procedure based on simulated-annealing (which includes another MILP model, a heuristic and a local search) have been designed and tested by means of a large computational experiment.

The AWALBP-L1-1 instances with few precedence relationships (1,200 PCB instances) are solved very efficiently, regardless of other characteristics such as workpiece length, number of workstations and number of tasks; specifically, 99.5% proven optimal solutions are obtained with our solution scheme.

The difficulty of the problem increases with the number of precedence relationships. However, our procedure still performs well for the 3,600 general instances. The average gap between the solutions found and the best calculated lower bound of the general instances with less than 200 tasks is 5.58%. The average gap of the other general instances is greater but the computational experiments show that this may be because the lower bound is less accurate.

In the design of the procedure for solving the problem not only the quality of the solutions but also the limitations concerning computing time have been taken into account, because the prospects are to use or adapt pre-processing and matheuristic procedures as components of algorithms for solving AWALBP-L1-M and upper levels of the AWALBP.

Concerning the AWALBP-L1 with precedence constraints, we envisage several research lines. Other metaheuristics different to SA (such as tabu search, genetic algorithms or particle swarm optimization) are worth exploring. On the other hand, the present work can be extended to AWALBP-L1-M; for instance, another step can be introduced in the proposed matheuristic in which tasks are assigned to workstations. Another line of research is the solution of AWALBP-L1 with a mixed-model line together with the model sequencing problem that may emerge.

ACKNOWLEDGEMENTS

The authors wish to express their gratitude to the anonymous reviewers for their valuable comments, which have helped to improve the quality of this paper.

REFERENCES

Abdullah Make, M.R., Ab. Rashid, M.F.F., Razali, M.M., 2017. A review of two-sided assembly line balancing problem. *International Journal of Advanced Manufacturing Technology* , 89, 1743-1763.

Adenso-Díaz, B., Laguna, M., 2006. Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research*, 54, 99-114.

Akpinar, S., Elmi, A., Bektaş, T., 2017. Combinatorial Benders cuts for assembly line balancing problems with setups. *European Journal of Operational Research*, 259, 527-537.

Battaïa, O., Dolgui, A., 2013. A taxonomy of line balancing problems and their solution approaches. *International Journal of Production Economics*, 142, 259-277.

Bautista, J., Pereira, J., 2011. Procedures for the time and space constrained assembly line balancing problem. *European Journal of Operational Research*, 212, 473-481.

Baybars, I., 1986. A survey of exact algorithms for the simple assembly line balancing problem. *Management Science*, 32, 909-932.

Becker, C., Scholl, A., 2006. A survey on problems and methods in generalized assembly line balancing. *European Journal of Operational Research*, 168, 694-715.

Bentaha, M.L., Battaïa, O., Dolgui, A., Hu, S.J., 2015. Second order conic approximation for disassembly line design with joint probabilistic constraints. *European Journal of Operational Research*, 247, 957-967.

Borba, L., Ritt, M., Miralles, C., 2018. Exact and heuristic methods for solving the robotic assembly line balancing problem. *European Journal of Operational Research*, doi: 10.1016/j.ejor.2018.03.011.

Bowman, E.H., 1960. Assembly-line balancing by linear programming. *Operations Research*, 8, 385-389.

Boysen, N., Fliedner, M., Scholl, A., 2007. A classification of assembly line balancing problems. *European Journal of Operational Research*, 183, 674-693.

Boysen, N., Fliedner, M., Scholl, A., 2008. Assembly line balancing: Which model to use when? *International Journal of Production Economics*, 111, 509-528.

Calleja, G., Corominas, A., García-Villoria, A., Pastor, R., 2013. A MILP model for the accessibility windows assembly line balancing problem (AWALBP). *International Journal of Production Research*, 51, 3549-3560.

Calleja, G., Corominas, A., García-Villoria, A., Pastor, R., 2014. Combining matheuristics and MILP to solve the accessibility windows assembly line balancing problem level 2 (AWALBP-L2). *Computers & Operations Research*, 48, 113-123.

Calleja, G., Corominas, A., García-Villoria, A., Pastor, R., 2016. Hybrid metaheuristics for the accessibility windows assembly line balancing problem level 2 (AWALBP-L2). *European Journal of Operational Research*, 250, 760-772.

Capacho, L., Pastor, R., Dolgui, A., Gunshinskaya, O., 2009. An evaluation of constructive heuristic methods for solving the alternative subgraphs assembly line balancing problem. *Journal of Heuristics*, 15, 109-132.

Fathi, M., Álvarez, M.J., Rodríguez, V., 2017. A new heuristic-based bi-objective simulated annealing method for U-shaped assembly line balancing. *European Journal of Industrial Engineering*, 10, 145-169.

Fleszar, K., 2017. A new MILP model for the accessibility windows assembly line balancing problem level 2 (AWALBP-L2). *European Journal of Operational Research*, 259, 169-174.

García-Villoria, A., Corominas, A. Pastor, R., 2012. Pure and hybrid metaheuristics for the response time variability problem. Chapter 10 in *Meta-Heuristics Optimization Algorithms in Engineering, Business, Economics, and Finance*, ed. P. Vasant, IGI Global.

García-Villoria, A., Corominas, A., Pastor, R., 2015. Heuristics and simulated annealing procedures for the accessibility windows assembly line balancing problem level 1 (AWALBP-L1). *Computers & Operations Research*, 62, 1-11.

Gaudlitz, R., 2004. Optimization algorithms for complex mounting machines in PC board manufacturing. *Diploma thesis*, Technical University of Darmstadt, Germany.

Koltai,T., Kalló, N., 2017. Analysis of the effect of learning on the throughput-time in simple assembly lines. *Computers and Industrial Engineering*, 111, 507-515.

Krishnan, K.K., Almaktoom, A.T., Udayakumar, P., 2016. Optimisation of stochastic assembly line for balancing under high variability. *International Journal of Industrial and Systems Engineering*, 22, 440-465.

Martin, R., 2002. Modeling an optimization problem from the automated manufacting of PC boards. *Diploma thesis*, Universität Konstanz, Germany.

Moreira, M.C.O., Cordeau, J-F., Costa, A.M., Laporte, G., 2015. Robust assembly line balancing with heterogeneous workers. *Computers & Industrial Engineering*, 88, 254-263.

Morrison, D.R., Sewell, E.C., Jacobson, S.H., 2014. An application of the branch, bound, and remember algorithm to a new simple assembly line balancing dataset. *European Journal of Operational Research*, 236, 403-409.

Müller-Hannemann, M., Weihe, K., 2006. Moving policies in cyclic assembly-line scheduling. *Theoretical Computer Science*, 351, 425-436.

Nikolaev, A.G., Jacobson, S.H., 2010. Simulated Annealing. Chapter 1 in *Handbook of Metaheuristics*, Eds. Gendreau and Potvin, Springer, 2nd edition.

Nowicki, E., Smutnicki, C., 1996. A fast taboo search algorithm for the job shop problem. *Management of Science*, 42, 797-813.

Otto, A., Battaïa, O., 2017. Reducing physical ergonomic risks at assembly lines by line balancing and job rotation: A survey. *Computers and Industrial Engineering*, 111, 467-480.

Quyen, N.T.P., Chen, J.C., Yang, C.-L., 2017. Hybrid genetic algorithm to solve resource constrained assembly line balancing problem in footwear manufacturing. *Soft Computing*, 21, 6279-6295.

Ritt, M., Costa, A.M., 2015. Improved integer programming models for simple assembly line balancing and related problems. *International Transactions in Operational Research*, 25, 1345-1359.

Roy, B., Saussmann, B., 1964. Les problèmes d'ordonnancement avec contraintes disjonctives. Note DS No. 9 bis, SEMA, Paris.

Scholl, A., Becker, C., 2006. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research*, 168, 666-693.

Sewell, E.C., Jacobson, S.H., 2012. A branch, bound, and remember algorithm for the simple assembly line balancing problem. *INFORMS Journal on Computing*, 24, 433-442.

Stille, W., 2008. Solution techniques for specific bin packing problems with applications to assembly line optimization. *PhD thesis*, Technical University of Darmstadt, Germany.

Tazari, S., 2006. Algorithmic approaches for two fundamental optimization problems: workload-balancing and planar steiner trees. *Diploma thesis*, Technical University of Darmstadt, Germany.

Zelter, L., Aghezzaf, E.-H., Limère, V., 2017. Workload balancing and manufacturing complexity levelling in mixed-model assembly lines. *International Journal of Production Research*, 55, 2829-2844.