Heuristics and simulated annealing procedures for the accessibility windows assembly line problem level 1 (AWALBP-L1)

Alberto GARCÍA-VILLORIA*, Albert COROMINAS and Rafael PASTOR
Institute of Industrial and Control Engineering (IOC)
Universitat Politècnica de Catalunya (UPC)
{alberto.garcia-villoria / albert.corominas / rafael.pastor}@upc.edu

Abstract. In some assembly lines, the workpieces are larger than the workstations. This implies that at a given instant the workstations have access to only a portion of the workpieces. In this context, the accessibility windows assembly line balancing problem (AWALBP) arises. In the AWALBP, 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. In each stationary stage, the workstations have access to different parts of the workpieces. This work solves the first level of AWALBP (AWALBP-L1), which consists in assigning the tasks among the workstations and stationary stages. Specifically, it is considered the AWALBP-L1 case in which the tasks can be processed in several workstations and their processing times depend on the workstation in which the tasks are processed. To solve the problem, we propose several heuristics and simulated annealing procedures. An extensive computational experiment is carried out to evaluate their performance.

Keywords: assembly line balancing, accessibility windows

1. Introduction

Assembly lines are of great importance in mass production systems, such as in the automotive and printed circuit board (PCB) industries. An assembly line consists of a serially organized set of workstations in which the product flows and a group of assembly operations (tasks) are performed in a limited duration (cycle time). The core of assembly line balancing problems (ALBPs) consists in assigning the tasks to workstations in order to optimise one or multiple efficiency objectives while satisfying some specific conditions. Because of the practical importance of this family of problems and the difficulty to solve them optimally (they are NP-hard; see, e.g., Wee and Magazine, 1982), a continuous research activity has been done over the last decades. Recent surveys are Becker and Scholl (2006), Scholl and Becker (2006), Boysen *et al*. (2007, 2008) and Battaïa and Dolgui (2013).

Among these problems, the simple ALBP (SALBP) has been traditionally the most studied (Baybars, 1986). SALBP is based on the following assumptions (Boysen *et al*., 2007): i) production of one homogeneous product, ii) all tasks are processed in a predetermined mode, iii) paced line with a fixed common cycle time, iv) one line which is serial and with no feeder lines or parallel elements, v) the processing sequence of tasks is subject to precedence relationships, vi) the task times are deterministic, vii) no assignment restrictions of tasks (besides precedence constraints), viii) a task cannot be split among two or more stations, and ix) all stations are equally equipped and have full

* 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 40107024; e-mail: alberto.garcia-villoria@upc.edu

access to the product piece. Because the assumptions of SALBP usually are restricting, researchers have intensified their efforts examining further aspects of real systems in order to address more realistic problems. For instance, among others: parallel workstations (Lusa, 2008), parallel tasks (Inman and Leon, 1994), multiple products (Pastor *et al*., 2002), mixed-models (Ding *et al*., 2006), U-shaped lines (Miltenburg, 2002), stochastic task times (Gamberini *et al*., 2006), setup times between tasks (Martino and Pastor, 2010), task times depending on the sequence (Capacho *et al*., 2009), incompatibility between tasks (Park *et al*., 1997), constrained resources (Corominas *et al*., 2011) and ergonomics considerations (Cheshmehgaz *et al*., 2012**)**.

It is usually assumed in the ALBP literature that at any moment each workstation has full access to one workpiece and each workpiece is being performed by only one workstation. However, in some contexts 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. Thus, at a given instant, one workpiece may be performed by several workstations and one workstation may perform tasks of one workpiece or two consecutive workpieces (see Figure 1). The presence of this characteristic gives rise to the problem named accessibility windows assembly line balancing problem (AWALBP) (Calleja et al., 2013).

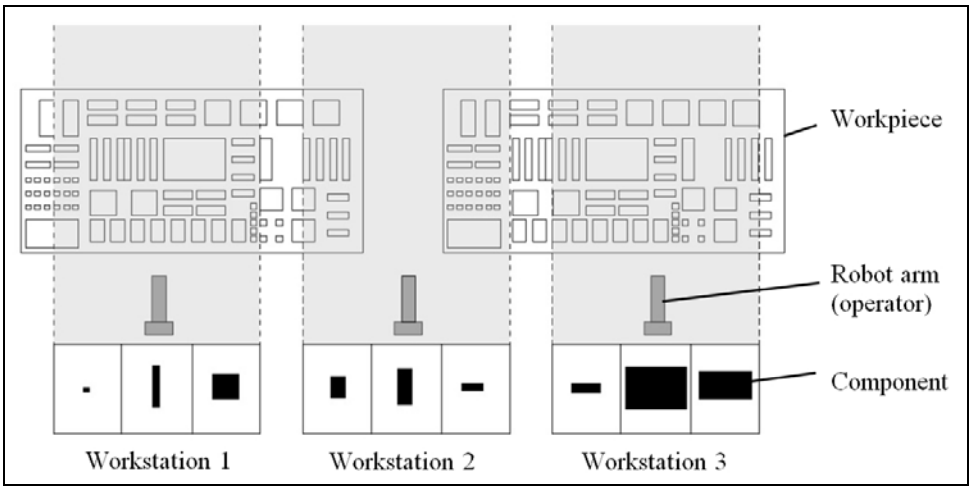

Figure 1. Example of an assembly line with accessibility windows (grey areas)

The cycle in the AWALBP splits into several *stationary stages* separated by *forward steps*. During a stationary stage, the line stands motionless and the workstations can perform the tasks that are accessible in their accessibility windows. During a forward step the line (together with the workpieces on it) moves forward and, thus, new tasks are accessible to the workstations. No tasks can be performed during the forward steps. The lengths of the movements may be different but must be multiple of a constant Δ that depends on the technology of the assembly line. We adopt Δ as the length unit.

Figure 2 has four snapshots of a cycle split into three stationary stages, where δ_k is the length of the forward step k ($k = 1,...,3$). The snapshots show the positions of the workpieces during the stationary stage k. In the first stationary stage of a cycle, these positions are determined by an initial shift x_0, which is the distance of the right border of the first workpiece with respect to the reference point 0 (which coincides with the left

border of the first workstation). Moreover, note that each workpiece takes over the position of its immediate predecessor in the next cycle.

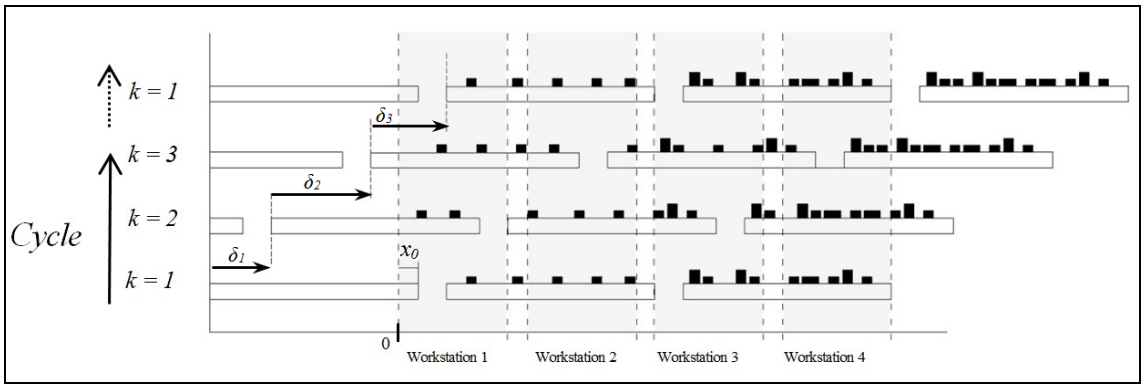

Figure 2. Example of a cycle split into three stationary stages (the grey areas indicate the accessibility windows of the workstations)

AWALBP arises in the throughput optimization of some highly automated assembly lines. For example, it can be found in the manufacturing of PCBs (Müller-Hannemann and Weihe, 2006) on specific robotic lines. On the PCBs, a number of electronic components are located at predefined positions and wired through electrical connections. In recent years, the size of electronic devices has been drastically reduced due to more compact circuit design, the introduction of surface-mount technology and the miniaturization of electronic components. As a result, automated machines are required which are capable of placing small-sized surface-mount devices with the necessary precision onto the board (Kulak *et al*., 2008). This kind of lines consists of several robots (workstations) clustered together. In this application, the dimensions of the boards are larger than the workstations; thus this scenario corresponds to an accessibility windows problem. Since most of today's technical products utilise electronic control units which include PCBs as key elements, the solution of the AWALBP is relevant in real life. For instance, several authors have been collaborated with Philips/Assembléon in the research, from a theoretical and a practical point of view, on the optimization of its component mounting assembly lines of PCBs (Martin, 2002; Gaudlitz, 2004; Tazari, 2006; Stille, 2008).

The AWALBP is a complex problem that involves several decisions. These decisions can be arranged according to tactical and operational decisions by the following four levels (Calleja *et al*., 2013):

- L1. This level consists in the tasks assignment; that is, to assign each task to be performed in one workstation during one stationary stage.
- L2. This level consists in the movement scheme; that is, to set the initial shift x_0, the number of stationary stages and the lengths of the forward steps.
- L3. This level consists in the workstation configuration; that is, to set the types of components, feeders, toolbits, etc. available at each workstation.
- L4. This level consists in the line configuration; that is, to set the number of workstations, technology of the line, etc.

Four variants of the AWALBP can be defined: AWALBP-L1 to AWALBP-L4. Each variant implies to solve its associated level decisions and the precedent ones. For

instance, the AWALBP-L2 consists of deciding the movement scheme and the tasks assignment when the workstation and line configurations are given. A more detailed description of the AWALBP can be found in Calleja *et al*. (2013).

Relatively few studies in the literature of assembly lines consider the presence of accessibility windows. Most of the related work has been inspired by throughput optimization in PCBs lines and several procedures have been proposed to solve the levels L2 (Tazari, 2006; Stille, 2008) and L3 (Martin, 2002; Gaudlitz, 2004). However, these works were developed for the specific problem faced by Philips/Assembléon. More general approaches to solve the AWALBP-L2 are proposed in Müller-Hannemann and Weihe (2006) and Calleja *et al*. (2013, 2014) but always under the assumption that each task can be processed in only one workstation.

AWALBP-L1 procedures have been proposed as an embedded step of main procedures to solve AWALBP-L2. Müller-Hannemann and Weihe (2006) solve AWALBP-L1 heuristically and Calleja *et al*. (2014) propose an efficient mathematical model called *Task model* to solve the problem optimally (again, for the case that each task can be processed in only one workstation); both works assume that each task can be processed in only one workstation. Tazari (2006) and Tazari *et al*. (2006) deal a case of AWALBP-L1 in which each task can be processed in several workstations. In addition to this characteristic, we consider also processing times depending in which workstation the task is processed.

This work deals with an specific case of the AWALBP-L1, in which the main assumptions are the following: all workpieces are identical, each task can be performed in a subset of workstations, the processing times depends only on the task and in which workstation it is performed, and there are no precedence relationships between tasks. The objective function is to minimise the cycle time (which is equivalent to maximise the production throughput). To solve it, we propose 24 heuristic procedures and 4 simulated annealing (SA) procedures. The resolution of the problem is not only important by itself, but also as a component of procedures for solving higher levels of AWALBP when the tasks can be processed in more than one workstation. Thus, the quality of the solutions is important together with the computing time to obtain them.

The rest of this paper is organised as follows. Section 2 describes the problem dealt with in this work. Section 3 formalizes the problem with a mixed integer linear programming (MILP) model. Sections 4 and 5 introduce, respectively, heuristics and simulated annealing procedures to solve the problem. An extensive computational experiment is carried out and the obtained results are analyzed in Section 6. Some final remarks and future research lines are given in Section 7.

2. Problem statement

The AWALBP-L1 case dealt with in this work is defined as follows. A (potentially infinite) number of identical workpieces has to be processed in a serial assembly line. Let A_0 be the length of the workpieces and A ($A > A_0$) the distance between the right borders of two successive workpieces on the assembly line.

The assembly line has M workstations. The accessibility window of workstation i ($i = 1,...,M$) is determined by the range $[L_i, R_i]$ (the accessibility windows concerns only one dimension in which the line moves, see Figure 1 and 2), where $R_i > L_i$; without loss of generality, we assume 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, $\exists_{i=1..M} A_0 > R_i - L_i$. The workstations have different configurations and for each workstation i it is known the set of tasks J_i that can perform.

There are N ($N = \left| \bigcup_{i=1..M} J_i \right|$) tasks to perform on each workpiece. There are no precedence constraints between tasks and they must be processed without preemption. For each task j ($j = 1,...,N$) it is known: the set of workstations I_j in which it can be processed ($I_j = \{i = 1,...,M : j \in J_i\}$), the processing time p_{ij} to perform task j in workstation $i \in I_j$, and the distance a_j to the right border of the workpiece in which the task must be performed.

The cyclic movement of the workpieces is determined by a given *movement scheme*. It is defined by: i) the initial shift x_0, corresponding to the distance of the right border of the first workpiece with respect to the reference point 0, ii) the number S of stationary stages in which the tasks can be performed, and iii) the length $\delta_k \in \mathbb{N}^+$ of the forward step k ($k = 1,...,S$) in which the line moves in positive direction. The time to move the line in a cycle at the steady speed is T_0. Moreover, for each forward step it has also to be added a fixed time T to take into account acceleration and deceleration.

A movement scheme is feasible if the two following conditions are met: i) each workpiece has to take over the position of its immediate predecessor in the next cycle; that is, the sum of the lengths δ_k of all forward steps is equal to A, and ii) for each task there exists at least one stationary stage in which the task is accessible to one or more workstations that can process the task. From the above information, it is derived the set Π_{ij} of the stationary stages in which task j is accessible in workstation i ($i = 1,...,M$; $j \in J_i$).

A solution of the problem consists in assigning for each task the workstation and the stationary stage in which the task is performed. The objective is to minimise the cycle time, CT, which has two parts: i) the time to move the line equal to $T_0 + S \cdot T$, and ii) the time needed in the stationary stages to complete all tasks. Since the first part is a constant (recall that at the level L1 the movement scheme is given), it is not regarded for optimisation purposes and the equation of the objective function is $Z = \sum_{k=1}^{S} C_k$, where C_k is the minimum time needed in stationary stage k so all workstations can process their assigned tasks.

Thus, according to the classification proposed in Boysen *et al.* (2007), the problem to solve is classified as [spec, fix| |c].

3. MILP model

The following mixed integer linear programming (MILP) model represents the presented AWALBP-L1. The MILP uses a solution obtained previously in an heuristic way whose objective function value can be used as an upper bound. The model is formulated as follows:

Data

M	Number of workstations.
N	Number of tasks.
S	Number of stationary stages of the movement scheme.
J_i	Set of tasks that workstation i can perform ($i = 1,...,M$).
I_j	Set of workstations that can perform task j ($j = 1,...,N$): $I_j = \{i = 1,...,M : j \in J_i\}$.
p_{ij}	Processing time of task j in workstation i ($i = 1,...,M$, $j \in J_i$). Without loss of generality, it is assumed that they are integer.
Π_{ij}	Set of stationary stages in which task j is accessible by workstation i ($i = 1,...,M$; $j \in J_i$).
Sol_H, Z_H	Solution obtained heuristically (for instance, with any of the procedures proposed in Sections 4 and 5) and its value, respectively.

Variables

$y_{ijk} \in \{0,1\}$	1 if task j is assigned to workstation i and it is processed during stationary stage k ($i = 1,...,M$; $j \in J_i$; $k \in \Pi_{ij}$).
$C_k \geq 0$	Time needed in stationary stage k so all workstations can process their assigned tasks ($k = 1,...,S$).

Model

$$[\text{MIN}]\ Z = \sum_{k=1}^{S} C_k \tag{1}$$

$$\sum_{k=1}^{S} C_k \leq Z_H - 1 \tag{2}$$

$$\sum_{i \in I_j} \sum_{k \in \Pi_{ij}} y_{ijk} = 1 \qquad j = 1,...,N \tag{3}$$

$$C_k \geq \sum_{j \in J_i : k \in \Pi_{ij}} p_{ij} \cdot y_{ijk} \qquad i = 1,...,M\ ;\ k = 1,...,S \tag{4}$$

The objective function (1) is to minimise the total time needed to complete all tasks. Constraint (2) imposes that the returned solution is strictly better than Sol_H if it exists; otherwise, the model is infeasible and, therefore, Sol_H is optimal. Constraints (3) enforce that each task is assigned to one workstation and processed during one

stationary stage. Finally, constraints (4) impose that the completion time to perform all tasks during an stationary stage is not less than the workload time of any workstation in that stationary stage.

Because the processing times are integer, the optimal objective function value is also integer. This justifies constraint (2) and allows setting to 1-ε (where $\varepsilon = 10^{-6}$) the absolute gap when solving the model.

4. Heuristic procedures

The MILP model proposed in Section 3 is able to find optimal or feasible solutions only for small instances, even with 1 hour of computational time, as the computational experiment shows (Section 6). Therefore, we present several heuristic procedures having a common scheme composed by two phases. In the first phase (constructive phase) an initial solution is obtained. The second phase (improvement phase) aims to improve the initial solution previously obtained.

Sections 4.1 and 4.2 explain the general scheme of the constructive phase and the improvement phase, respectively. Section 4.3 proposes 24 heuristic procedures that share the aforementioned common scheme.

4.1. Constructive phase

To obtain a solution, each task has to be assigned to a workstation and to a stationary stage. The assignment of tasks to stationary stages given an assignment of tasks to workstations may be solved optimally very quickly (Calleja *et al*., 2014). Thus, the constructive phase does these assignments in two steps. In the first step, the tasks are assigned to workstations; in the second step, they are assigned to stationary stages.

4.1.1. Constructive phase, first step: assignment of tasks to workstations

The rationality behind the first step of the constructive phase is to assign the tasks to workstations so the maximum of their workloads is reduced and also to increase the number of possible feasible assignments of tasks to stationary stages in the second step of the constructive phase (see Section 4.1.2).

To solve the problem of assigning tasks to workstations, the following MILP model is used in which the values of the parameters α_1 and α_2 have to be set:

Variables

$y_{ij} \in \{0,1\}$ 1 if task j is assigned to workstation i ($i = 1,...,M$; $j \in J_i : \Pi_{ij} \neq \varnothing$).

$MW \geq 0$ Maximum workload time of the workstations.

Model

$$[\text{MIN}]\ Z' = \alpha_1 \cdot MW - \alpha_2 \cdot \sum_{i=1}^{M} \sum_{j \in J_i : \Pi_{ij} \neq \varnothing} \left|\Pi_{ij}\right| \cdot y_{ij} \qquad (5)$$

$$\sum_{i \in I_j : \Pi_{ij} \neq \emptyset} y_{ij} = 1 \qquad j = 1, ..., N \tag{6}$$

$$MW \geq \sum_{j \in J_i : \Pi_{ij} \neq \emptyset} p_{ij} \cdot y_{ij} \qquad i = 1, ..., M \tag{7}$$

The objective function (5) is to minimise the weighted difference of the maximum workload time (MW) and the total number of stationary stages in which the tasks are accessible to their assigned workstations ($\sum_{i=1}^{M} \sum_{j \in J_i : \Pi_{ij} \neq \emptyset} \left|\Pi_{ij}\right| \cdot y_{ij}$). Constraints (6) enforce that each task is assigned to one workstation. Finally, constraints (7) impose that the maximum workload time is not less than the workload time of any workstation.

We propose the following four alternatives for the values of the parameters α_1 and α_2:

- (A1). To minimise the maximum workload time of the workstations: $\alpha_1 = 1$, $\alpha_2 = 0$.
- (A2). To minimise the maximum workload time of the workstations using as tie breaker the largest total number of stationary stages in which the tasks are accessible to their assigned workstations: $\alpha_1 = 1$, $\alpha_2 = \frac{1}{\sum_{j=1}^{N} \left(\max_{i \in I_j} \left|\Pi_{ij}\right| \right) + 1}$.
- (A3). To maximise the total number of stationary stages in which the tasks are accessible to their assigned workstations: $\alpha_1 = 0$, $\alpha_2 = 1$.
- (A4). To maximise the total number of stationary stages in which the tasks are accessible to their assigned workstations using as tie breaker the smallest maximum workload time of the workstations: $\alpha_1 = \frac{1}{\max_{i=1}^{M} \left(\sum_{j \in J_i : \Pi_{ij} \neq \emptyset} p_{ij} \right) + 1}$, $\alpha_2 = 1$

If the alternative A1 or A3 is used to set the α_1 and α_2 values, then the absolute gap is set to 1-ε ($\varepsilon = 10^{-6}$); otherwise, the absolute gap is set to 0.

4.1.2. Constructive phase, second step: assignment of tasks to stationary stages

Once the first step of the constructive phase has been applied and the tasks have been assigned to one workstation, the following data can be derived. Let $\hat{y}_{ij}$ be the value of variable y_{ij} obtained after solving the model of Section 4.1.1. Thus, we define:

W_j The workstation in which task j has been assigned ($j = 1, ..., N$): $W_j = \sum_{i \in I_j : \Pi_{ij} \neq \emptyset} i \cdot \hat{y}_{ij}$.

TA_i The set of tasks assigned to workstation i ($i = 1, ..., M$): $TA_i = \left\{ j = 1, ..., N : W_j = i \right\}$.

In the second step of the constructive phase, the tasks are assigned to one stationary stage (among the ones in which the tasks are accessible to the assigned workstation). To do the assignment, the *Task model* proposed in Calleja *et al*. (2014), which is very efficient, is used. The *Task model* is formulated as follows:

Variables

$y_{jk} \in \{0,1\}$ 1 if task j is assigned to stationary stage k ($j = 1,...,N$; $k \in \Pi_{W_j,j}$).

$C_k \geq 0$ Time to complete the tasks performed during stationary stage k ($k = 1,...,S$).

Model

$$[\text{MIN}]\ \text{Z} = \sum_{k=1}^{S} C_k \qquad (1)$$

$$\sum_{k \in \Pi_{W_j,j}} y_{jk} = 1 \qquad j = 1,...,N \qquad (8)$$

$$C_k \geq \sum_{j \in TA_i : k \in \Pi_{ij}} p_{ij} \cdot y_{jk} \qquad i = 1,...,M\ ;\ k = 1,...,S \qquad (9)$$

The objective function (1) is to minimise the total time needed to complete all tasks. Constraints (8) enforce that each task is assigned to one stationary stage. Finally, constraints (9) impose that the completion time of each stationary stage is not less than the workload time of any workstation during that stationary stage.

Again, the absolute gap is set to 1-ε ($\varepsilon = 10^{-6}$).

The solution obtained in the second step is a feasible solution of the overall problem and is taken as the solution of the constructive phase.

4.2. Improvement phase

The aim of the improvement phase is to improve the solution obtained in the constructive phase. We propose two alternative procedures to improve the initial solution: *IP1* and *IP2*.

4.2.1. Improvement procedure IP1

The cornerstone of *IP1* is iteratively improving the solution by changing the assignment of one task to another workstation and/or another stationary stage.

The algorithm of *IP1* is shown in Figure 3, where a solution is represented by the set $SOL = \{\Gamma_{ik}\}$ and Γ_{ik} is the set of tasks assigned to workstation i and to be performed during stationary stage k ($i = 1,...,M$; $k = 1,...,S$). Additionally, the following nomenclature is used in the description of the algorithm:

- wt_{ik} : workload time of workstation i in stationary stage k : $wt_{ik} = \sum_{j \in \Gamma_{ik}} p_{ij}$.

- $Z(SOL)$: Z value of solution SOL: $Z(SOL) = \sum_{k=1}^{S} \left(\max_{i=1}^{M} wt_{ik} \right)$.
- $\hat{i}_k^1$, $\hat{i}_k^2$: most and second most workloaded workstation, respectively, in stationary stage k: $\hat{i}_k^1 = \arg\max_{i=1,\ldots,M} wt_{ik}$, $\hat{i}_k^2 = \arg\max_{i=1,\ldots,M: i \neq \hat{i}_k^1} wt_{ik}$.
- $\Delta Z\left(j, \hat{k}, i, k\right)$: increase (positive or negative) of the Z value of the current solution if task $j \in \Gamma_{\hat{i}_k^1, \hat{k}}$ (i.e., the task j assigned currently to the most workloaded workstation in stationary stage $\hat{k}$) is assigned to workstation i and stationary stage k ($i \in I_j, k \in \Pi_{ij} : \hat{i}_{\hat{k}}^1 \neq i \vee \hat{k} \neq k$):

$$\Delta Z\left(j, \hat{k}, i, k\right) = \begin{Bmatrix} \max\left(wt_{\hat{i}_{\hat{k}}^1, \hat{k}} - p_{\hat{i}_k^1, j}, wt_{\hat{i}_{\hat{k}}^2, \hat{k}}, wt_{ik} + p_{ij}\right) - wt_{\hat{i}_k^1, \hat{k}} & \text{if } \hat{k} = k \\ \max\left(0, wt_{ik} + p_{ij} - wt_{\hat{i}_k^1, k}\right) - \min\left(p_{\hat{i}_k^1, j}, wt_{\hat{i}_k^1, \hat{k}} - wt_{\hat{i}_k^2, \hat{k}}\right) & \text{if } \hat{k} \neq k \end{Bmatrix}$$

- *getNeighbour*(SOL): function that returns the first neighbour that improves SOL if exists; otherwise, it returns SOL. Its algorithm is shown in Figure 4 (where ES and TS are the remaining set of stationary stages and the current remaining set of tasks, respectively, to examine to obtain neighbours).
- *getTask*(TS): function that returns a task of the set TS. The criterion to select the task is discussed below.

Sol^* = Apply the constructive phase; *improvement* = true
While (*improvement*) **do**:
  $SolN = getNeighbour\left(Sol^*\right)$
  $improvement = Z\left(SolN\right) < Z\left(Sol^*\right)$
  If (*improvement*) **then** $Sol^* := SolN$ **End if**
End while
Apply *Task model* (see Section 4.1.2) using the Sol^* assignments of tasks to workstations (but not the assignments to stationary stages) and return the obtained solution

---

Figure 3. *IP1* algorithm

$ES = \{1,...,S\}$; *improvement* = false

While ($\neg$*improvement* $\wedge$ $ES \neq \varnothing$) **do**:

$$\hat{k} = \arg\max_{k \in ES} \left(wt_{\hat{i}^1_k,k} - wt_{\hat{i}^2_k,k} \right);\ ES = ES \setminus \left\{ \hat{k} \right\};\ TS = \left\{ j \in \Gamma_{\hat{i}^1_k,\hat{k}} : \sum_{i \in I_j} \left| \Pi_{ij} \right| \geq 2 \right\}$$

While ($\neg$*improvement* $\wedge$ $TS \neq \varnothing$) **do**:

$$j^* = getTask(TS);\ TS = TS \setminus \left\{ j^* \right\}$$

Get the workstation i^* and stationary stage k^* that most would improve the solution if task j^* is assigned to i^* and k^*:

$$\left(i^*, k^* \right) = \arg\min_{i \in I_{j^*}, k \in \Pi_{ij^*}; \hat{i}^1_{\hat{k}} \neq i \vee \hat{k} \neq k} \left(\Delta Z \left(j^*, \hat{k}, i, k \right) \right)$$

$$improvement = \Delta Z \left(j^*, \hat{k}, i^*, k^* \right) < 0$$

If (*improvement*) **then** j^* is assigned to i^* and k^*; that is**:**

$$\Gamma'_{\hat{i}^1_{\hat{k}},\hat{k}} = \Gamma_{\hat{i}^1_{\hat{k}},\hat{k}} \setminus \left\{ j^* \right\},\ \Gamma'_{i^*k^*} = \Gamma_{i^*k^*} \cup \left\{ j^* \right\}$$

$$SolN = \left(SOL \setminus \left\{ \Gamma_{\hat{i}^1_{\hat{k}},\hat{k}}, \Gamma_{i^*k^*} \right\} \right) \cup \left\{ \Gamma'_{\hat{i}^1_{\hat{k}},\hat{k}}, \Gamma'_{i^*k^*} \right\}$$

End if

End while

End while

If (*improvement*) **then** return *SolN* **otherwise** return *SOL* **End if**

Figure 4. *getNeighbour*(*SOL*) function

We propose the following three alternatives as criterion to select the task in function $getTask(TS)$:

- (C1). The task with the largest process time: $getTask(TS) = \arg\max_{j \in TS} p_{\hat{i}^1_{\hat{k}},j}$.
- (C2). If there is a task with a process time not less than the difference between the first and second workload times (in stationary stage $\hat{k}$) then prioritize the task with the smallest processing time; otherwise, prioritize the task with the largest processing time: $getTask(TS) = \arg\max_{j \in TS} \left(\min \left(p_{\hat{i}^1_{\hat{k}},j}, wt_{\hat{i}^1_{\hat{k}},\hat{k}} - wt_{\hat{i}^2_{\hat{k}},\hat{k}} \right) \right)$ and use as tie breaker the task with the smallest process time.
- (C3). The task with the largest number of pairs (workstation, stationary stage) in which it can be assigned: $getTask(TS) = \arg\max_{j \in ST} \left(\sum_{i \in I_j} \left| \Pi_{ij} \right| \right)$.

4.2.2. Improvement procedure IP2

The improvement procedure *IP2* is an extension of *IP1*. *IP2* consists in applying iteratively *IP1* until no improvement is obtained. The algorithm is shown in Figure 5.

Sol^* = Apply the constructive phase; *improvement* = true
While (*improvement*) **do**:
$Sol' = IP1\left(Sol^*\right)$
$improvement = Z\left(Sol'\right) < Z\left(Sol^*\right)$
If (*improvement*) **then** $Sol^* = Sol'$ **End if**
End while
Return Sol^*

Figure 5. *IP2* algorithm

4.3. Proposed heuristics

We propose 24 heuristic procedures resulting from combining the 4 alternatives of the values of the parameters α_1 and α_2 in the first step of the constructive phase (A1 to A4, see Section 4.1.1), the 2 methods of the improvement phase (*IP1* and *IP2,* see Section 4.2) and the 3 criteria to define the function $getTask(\cdot)$ (C1 to C3, see Section 4.2.1). We name them *H_Ax_IPy_Cz* ($x = 1,...,4$; $y = 1,2$; $z = 1,2,3$), which refers to the heuristic with the alternative A*x*, the improvement procedure *IPy* and the criterion C*z*.

In all of them, the resolution of the two MILP models in the constructive phase are restricted, each one, to a maximum computing time of 10 seconds per instance. This restriction ensures that the initial solution is obtained in no more than 20 seconds. A preliminary computational experiment showed that the first model (the model to assign the tasks to workstations) is able to obtain optimal or near-optimal solutions in less than 10 seconds. With respect to the second model (*Task model*), Calleja *et al*. (2014) reported average computing times of few miliseconds, so we expect that the 10 seconds limit will rarely be active in practice (in our computational experiment, we confirmed the assumption).

5. Simulated annealing procedures

The simulated annealing metaheuristic (SA) has been successfully applied to solve a wide range of combinatorial optimisation problems (Nikolaev and Jacobson, 2010). In particular, it is one of the most popular metaheuristics for solving assembly line balancing problems (Battaïa and Dolgui, 2013).

SA starts from a solution, which is initially the current solution (constructive phase). Then, at each iteration, a new solution selected at random from the neighbourhood of the current solution is considered (improvement phase). Moves to non-improving solutions are allowed with a certain probability in order to avoid being trapped into a local optimum. Here we propose 4 SA-based procedures resulting from the 4 alternatives to construct the initial solution, A1 to A4 (see Section 4.1); we name them as *SA_Ax* ($x = 1,...,4$), respectively. Their pseudocode is shown in Figure 6 and the following additional nomenclature is used in the description of the algorithm:

- *getRandomNeighbour*(SOL): function that returns a neighbour of SOL selected at random. Its algorithm is shown in Figure 7

Sol^* = Apply the constructive phase with alternative A1, A2, A3 or A4
Set the values of parameters:
t_0 (initial temperature)
itt (number of iterations during the temperature remains equal)
β (cooling factor)
mni (maximum number of iterations without improving the best solution)
$SolC = Sol^*;\ t = t_0;\ ni = 0$
While ($ni < mni$) **do**:
$i = 0$
While ($(i < itt) \wedge (ni < nmi)$) **do**:
$SolN = getRandomNeighbour(SolC)$
$\Delta = Z(SolN) - Z(SolC)$
If ($\Delta < 0$) **then** $SolC = SolN$
otherwise $SolC = SolN$ with probability $\exp(-\Delta/t)$
End if
If ($Z(SolC) < Z(Sol^*)$) **then** $Sol^* = SolC;\ ni = 0$
otherwise $ni = ni + 1$
End if
$i = i + 1$
End while
$t = t \cdot \beta$
End while
Apply *Task model* (see Section 4.1.2) using the Sol^* assignments of tasks to workstations (but not the assignments to stationary stages) and return the obtained solution

---

Figure 6. SA algorithm

$\hat{k}$ = select uniformly at random from $\{1,...,S\}$
j^* = select uniformly at random from $\left\{ j \in \Gamma_{\hat{i}_{\hat{k}}^1,\hat{k}} : \sum_{i \in I_j} \left|\Pi_{ij}\right| \geq 2 \right\}$
(i^*, k^*) = select uniformly at random from $\left\{(i,k) \mid i \in I_{j^*}, k \in \Pi_{ij^*}\right\} \setminus \left\{\left(\hat{i}_{\hat{k}}^1, \hat{k}\right)\right\}$
$\Gamma'_{\hat{i}_{\hat{k}}^1,\hat{k}} = \Gamma_{\hat{i}_{\hat{k}}^1,\hat{k}} \setminus \{j^*\},\ \Gamma'_{i^*k^*} = \Gamma_{i^*k^*} \cup \{j^*\}$
Return $\left(SOL \setminus \left\{\Gamma_{\hat{i}_{\hat{k}}^1,\hat{k}}, \Gamma_{i^*k^*}\right\}\right) \cup \left\{\Gamma'_{\hat{i}_{\hat{k}}^1,\hat{k}}, \Gamma'_{i^*k^*}\right\}$

Figure 7. *getRandomNeighbour*(*SOL*) function

The SA procedures have 4 parameters (t_0, itt, β and mni, see Figure 6). The parameter mni is set to 50,000 since it is a large enough value for the convergence of the procedures. To set the remaining parameters, we used an automatic tool known as CALIBRA (Adenso-Díaz and Laguna, 2006), which is specifically designed for fine-tuning the parameters of algorithms. It is based on using conjointly Taguchi's fractional factorial experimental designs and a local search procedure. We applied CALIBRA to a training set of 96 instances generated in the same way as the test instances (explained in Section 6.1). The obtained parameters values are given in Table 1:

	SA_A1	***SA_A2***	***SA_A3***	***SA_A4***
t_0	62	58	58	48
itt	650	650	690	760
β	0.99	0.99	0.99	0.99

Table 1. Calibration of the SA-based procedures

6. Computational experiment

For our computational testing, we solved the MILP models using IBM ILOG CPLEX 12.6. Regarding the heuristic and SA procedures, we implemented them in Java SE 1.6.21. The experiments were run on a PC 3.33 GHz Pentium Intel Core i5 with 4 GB RAM.

We first explain how the test instances were generated. Then we compare the proposed procedures between them and later we compare the obtained solutions with known optimal solutions obtained with the mathematical model proposed in Section 3.

6.1. Test instances

We generated a set of test instances based on those 1200 instances of AWALBP-L2 used in Calleja *et al*. (2013, 2014), which are based on the description of real-world cases. According to Gaudlitz (2004), the workpiece length may be up to 2.5 times larger than the width of the workstations, the number of machines may range between 7 and 20, and the number of tasks between 100 and 800. The data that define each of the 1200 instances is summarized in Table 2, where $U[\cdot]$ refers to the discrete uniform distribution (for more details, see Calleja *et al*., 2013).

Name	Description	Value
A_0	Length of the workpieces	$U[11,40]$
A	Distance between the right borders of two successive workpieces	$A = A_0 + 1$
M	Number of workstations	$U[5,40]$
L_i	Left border of the accessibility window of workstation i ($i = 1,...,M$)	$L_i = 11 \cdot (i-1)$
R_i	Right border of the accessibility window of workstation i ($i = 1,...,M$)	$R_i = L_i + 10$
N	Number of tasks	$U[50,1000]$
WI_j	Workstation in which task j ($j = 1,...,N$) can be performed	$U[1,M]$
a_j	Distance of task j ($j = 1,...,N$) to the right border of the workpiece	$U[0,A_0]$
p_j	Processing time of task j ($j = 1,...,N$)	$U[100,150]$
T	Time to take into account acceleration and deceleration in forward steps	200

Table 2. Data of the AWALBP-L2 instances used in Calleja *et al*. (2013, 2014)

Recall that whereas in the AWALBP-L2 solved in Calleja *et al*. (2013, 2014) it is assumed that each task can be performed only in one workstation, in this work the tasks can be performed in multiple workstations. From each one of the existing 1200 instances, 2 new test instances are generated under the following two scenarios: low versatile tasks and high versatile tasks (*versatility* here qualifies the number of workstations in which a task can be processed). The values of A_0, A, M, L_i, R_i, N, a_j and T in each pair of new instances are equal to their original instance. The remaining data to generate are the workstations in which each task can be processed, the processing times and the movement scheme, which are derived as follows.

First, for each task j, the number of workstations that can process it, NW_j, is generated at random uniformly between 1 and $\max\left(2,\lceil 0.15 \cdot M \rceil\right)$ for the low versatile instances, and between 1 and $\max\left(2,\lceil 0.3 \cdot M \rceil\right)$ for the high versatile instances. The workstations that can process task j are WI_j (that is, the only workstation that could process task j in the original instance) together with, if $NW_j > 1$, $NW_j - 1$ workstations (excluding WI_j) selected at random.

To generate the processing times of each task j in the workstations that can process it, the workstations are classified in slow, normal and quick workstations. The type of each workstation is set at random (with equal probability for each type). Let PT_j^s, PT_j^n and PT_j^q be the process time of task j in a slow, normal and quick workstation, respectively. According to the type of the workstation, the process times are defined as

follows (where p_j is the process time of task j in the original instance and $\lceil x \rfloor = \lfloor x + 0.5 \rfloor$):

- If WI_j is slow then $PT_j^s = p_j$, $PT_j^n = \left\lceil \frac{PT_j^s}{1.05} \right\rfloor$, $PT_j^q = \lceil 0.95 \cdot PT_j^n \rfloor$
- If WI_j is normal then $PT_j^n = p_j$, $PT_j^s = \lceil 1.05 \cdot PT_j^n \rfloor$, $PT_j^q = \lceil 0.95 \cdot PT_j^n \rfloor$
- If WI_j is quick then $PT_j^q = p_j$, $PT_j^n = \left\lceil \frac{PT_j^q}{0.95} \right\rfloor$, $PT_j^s = \lceil 1.05 \cdot PT_j^n \rfloor$

Finally, the movement scheme is chosen among 10 movement schemes generated at random as follows. The initial shift x_0 is a uniformly distributed random value between its possible minimum value (0) and its maximum possible value ($\min\left(R_1 + \min_{j \in J_1} a_j, A-1\right)$, see Calleja *et al*., 2013). The number of forward steps S and their lengths δ_k are set at random as follows. Iteratively, until $\sum_{k=1}^{S} \delta_k = A$, the length of the current forward step k is a uniformly distributed random value between 1 and the minimum of these two values: i) $A - \sum_{l=1}^{k-1} \delta_l$, and ii) the maximum length of δ_k so each task j can be accessible in at least one workstation $i \in I_j$. Among the 10 random movement schemes, it is chosen the one with the lowest S value and as tie breaker the one less *irregular* (quantified as $\sum_{k=1}^{S} \left(\delta_k - \frac{A}{S} \right)^2$). The motivation to select a movement scheme with a low number of stationary stages is to reduce the acceleration and deceleration times and, additionally, it favours evenly balanced workloads in each stationary stage.

The set of the 2400 instances is available at https://www.ioc.upc.edu/EOLI/research/AWALBP_L1.

6.2. Results of the procedures

We analyze the performance of the proposed procedures. Table 3 shows, for the 24 heuristics and the 4 SA procedures, the average Z values of the initial constructed solution and the final improved solution together with the computing time (of each phase and the total time, in seconds).

	Constructive phase (Initial solution)		Improvement phase (Final solution)		Total time (s)
	Z	**Time (s)**	***Z***	**Time (s)**	
H_A1_IP1_C1	5024.67	10.65	4614.41	2.18	12.83
H_A1_IP1_C2			4639.92	2.09	12.74
H_A1_IP1_C3			4627.64	2.14	12.79
H_A1_IP2_C1			4555.04	5.79	16.44
H_A1_IP2_C2			4570.40	6.12	16.77
H_A1_IP2_C3			4564.62	6.10	16.75
SA_A1			4478.60	5.68	16.33
H_A2_IP1_C1	4885.85	11.94	4586.31	3.02	14.96
H_A2_IP1_C2			4605.22	2.99	14.93
H_A2_IP1_C3			4594.98	3.00	14.94
H_A2_IP2_C1			4533.29	7.97	19.91
H_A2_IP2_C2			4542.53	7.84	19.78
H_A2_IP2_C3			4538.50	7.95	19.89
SA_A2			4474.83	4.26	16.20
H_A3_IP1_C1	7307.63	0.49	5143.20	1.13	1.62
H_A3_IP1_C2			5502.84	0.97	1.46
H_A3_IP1_C3			5283.58	0.97	1.46
H_A3_IP2_C1			4719.24	11.68	12.17
H_A3_IP2_C2			4879.26	13.04	13.53
H_A3_IP2_C3			4810.86	12.06	12.55
SA_A3			4487.92	4.78	5.27
H_A4_IP1_C1	4852.05	8.97	4599.56	3.10	12.07
H_A4_IP1_C2			4618.90	3.00	11.97
H_A4_IP1_C3			4609.64	3.08	12.05
H_A4_IP2_C1			4543.35	8.20	17.07
H_A4_IP2_C2			4554.34	8.31	17.28
H_A4_IP2_C3			4548.17	8.54	17.51
SA_A4			4475.91	5.86	14.83

Table 3. Results of the heuristics and SA procedures.

6.2.1. Results of the heuristics

We analyse the influence of the components of the heuristics. We can see that the values of the parameters α_1 and α_2 have a great influence of the quality of the initial solution (and, indirectly, of the final solution). The best option, on average, for the quality of the initial solution is alternative A4 (that is, to maximise the total number of pairs (workstation, stationary stage)) in which the tasks can be performed using as tie breaker the smallest maximum workload time of the workstations. With respect to the computing time to generate the initial solution, alternatives A1, A2 and A4 are on average quite similar (although A4 is a bit faster than the other two alternatives). These three alternatives need around 10 s: usually most of the computing time in the constructive phase is spent on the first step whereas the second step most of the times is

very quick (less than 1 s) and solved optimally. On the other hand, A3 is around 20 times faster but the quality of the initial solutions are more than 30% worse than the quality obtained with the other alternatives.

With respect to the improvement phase, obviously *IP2* obtains better solutions than *IP1* (recall that *IP2* consists in applying iteratively *IP1*). When *IP2* is used instead of *IP1*, the quality of the solutions improves between 1.16% and 1.5% with alternatives A1, A2 and A4, and between 8.24% and 11.33% with alternative A3. And the computing time increases around 1.4 times (around 4 or 5 s) with A1, A2 and A4, and between 7.5 and 9.3 times (around 11 s) with A3.

With respect to the criterion to select the task in the function $getTask(\cdot)$, although C1 obtains the best quality averages, all three options obtains similar quality solutions with alternatives A1, A2 and A4 (differences smaller than 0.55%); larger differences are found when using A3: for instance, the difference between C2 and C1 is 3.28%.

We also observe that the best final solutions are not necessarily obtained with the best initial solutions. The best initial solutions, on average, are obtained when using alternative A4 but the best final solutions are obtained with A2.

The heuristic that finds best solutions on average is *H_A2_IP2_C1* with an average time of 19.91 s. On the other hand, one of the quickest heuristics is *H_A3_IP1_C1* which needs an average time of 1.62 s, but their solutions are 10.83% worse than the *H_A2_IP2_C1* solutions.

6.2.2. Results of the SA procedures

The 4 SA procedures obtain similar average Z values. The best average (obtained with *SA_A2*) is only 0.29% better than the worst average (obtained with *SA_A3*). Thus, the quality of the final solution returned by the SA procedures are quite independent of the quality of their initial solution (recall that the initial solution obtained with alternative A3 is more than 30% worse than the initial solutions obtained with the other alternatives). With respect to the computing time, procedures *SA_A1*, *SA_A2* and *SA_A4* spent around 15 s whereas *SA_A3* spends around 5 s. Although all 4 procedures has a similar computing time in the improvement phase, the constructive phase of *SA_A3* is very quick (close to 0.5 s).

When comparing the SA procedures versus the heuristics, we can see that all average Z values obtained with the SA procedures are better than the best heuristic average (obtained with *H_A2_IP2_C1*). Moreover, the average computing times of the SA procedures are lower.

Thus, we will focus on the behaviour of the SA procedures. Table 4 shows the average Z value (Z), average and maximum relative gap (in %) of the Z values with respect to the Z value of the best solution found by any of the heuristics or the SA procedures ($Gap\%$ and $MaxGap\%$, respectively) and the percentage of times that the best known solution is found ($Best\%$). More detailed results of *SA_A2*, the procedure that has the best average Z and $Gap\%$ values, are given in Appendix)

	SA_A1	*SA_A2*	*SA_A3*	*SA_A4*
Z	4478.60	4474.83	4487.92	4475.91
Gap%	0.71	0.57	0.92	0.61
MaxGap%	12.76	28.34	21.96	16.64
Best%	26.79	27.33	18.04	23.21

Table 4. Results of the SA procedures

SA_A1 and *SA_A2* are the procedures that most times find the best known solution although these number of times is low (less than 28%). However, all SA procedures has an average relative gap value less than 1%, which points that on average the obtained solutions are very near to the best known solutions. Specifically, *SA_A2* is the procedure that performs best on average. On the other hand, *SA_A2* presents the worst maximum relative gap and, instead, *SA_A1* has the lowest maximum relative difference.

6.3. Comparison with known optimal solutions

In order to obtain optimal solutions, the MILP model introduced in Section 3 was run with a run limit of 1 hour per instance and the procedure *SA_A2* was used to obtain the (heuristic) solution Sol_H . We tried to solve the 400 instances with the smallest length of the workpiece (the instances with $11 \le A_0 \le 15$); for larger instances, CPLEX could not find even a feasible solution in most cases. Table 5 shows the number of times that the model found a solution better than Sol_H distinguishing between proven optimal solutions and feasible solutions (excluding the proven optimal ones). Table 5 also shows the average relative gap of Sol_H with respect to the solution returned by the model. If the model does not find any solution (recall that constraint (3) ensures that the returned solution, if any, is strictly better than Sol_H) then Sol_H is considered as the solution returned by the model.

#Sol_H improved		**#Sol_H non improved**	
#Optimal	**#Feasible**	**#Optimal**	**#Feasible**
15 (7.26 %)	36 (6.74%)	13 (0 %)	336 (0%)

Table 5. Results of the MILP model (average gap of Sol_H between parenthesis).

We can see that 28 optimal solutions are obtained, which is only 7% of those 400 instances. For 13 of these 28 instances, *SA_A2* obtains an optimal solution. For the other 15 instances in which the model finds a solutions better than Sol_H , the average gap is 7.26%. However, the average computing time spent by the SA procedure is 3.20 s whereas the average time of the model is 168.11 s.

Regarding the feasible solutions, the model does not find (within 1 hour) a solution better than Sol_H for 336 instances and only for 36 instances the model finds a better solution. The average gap for these 36 instances is 6.74% but again the average time of the procedure, 12.19 s, is much less than the average time of the model (3600 s).

7. Conclusions

This paper deals with the AWALBP, a problem that may appear when the workpieces are larger than the width of the workstations. Thus, only a portion of the workpieces is accessible to the workstations at any moment. Specifically, we address the level L1 of this problem, in which the tasks have to be assigned to a workstation and a stationary stage and the processing times depends on the workstation in which the tasks are processed.

To solve this problem, we propose 24 heuristics and 4 SA procedures based on the heuristics. The aim when solving this problem is not only to obtain good solutions but also to do it in a low computing time, since these procedures may be used as components of future procedures to solve AWALBP-L2.

The computational experiment shows that the problem is very hard to solve optimally with the MILP model even for the instances with the smallest workpiece length. The procedure that obtains, on average, the best solutions is *SA_A2*. On the other hand, on average, the quality of the solutions of *SA_A3* are close but it requires 3 times less computing time. When compared the solutions obtained with *SA_A2* with the known optimal solutions, we detected that for 15 instances *SA_A2* does not obtain the optimal solution and the average gap is 7.26%.

Future research will be take into account more characteristics as, for instance, setup times to exchange the toolbit necessary for each task (e.g., Tazari, 2006).

REFERENCES

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

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

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.

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.

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.

Cheshmehgaz, H.R., Haron, H., Kazemipour, F., Desa, M.I., 2012. Accumulated risk of body postures in assembly line balancing problem and modelling through a multi-criteria fuzzy-genetic algorithm. *Computers & Industrial Engineering*, 63, 503-512.

Corominas, A., Ferrer, L., Pastor, R., 2011. Assembly line balancing: general resource-constrained case. *International Journal of Production Research*, 49, 3527-3542.

Ding, F-Y., Zhu, J., Sun, H., 2006. Comparing two weighted approaches for sequencing mixed-model assembly lines with multiple objectives. *International Journal of Production Economics*, 102, 108-131.

Gamberini, R., Grassi, A., Rimini, B., 2006. A new multi-objective heuristic algorithm for solving the stochastic assembly line re-balancing problem. *International Journal of Production Economics*, 102, 226-243.

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

Inman, R.R., Leon, M., 1994. Scheduling duplicate serial stations in transfer lines. *International Journal of Production Research*, 32, 2631-2644.

Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P., 1983. Optimization by simulated annealing, *Science*, 220, 671-680.

Kulak, O., Yilmaz, I.O., Günther, H.-O., 2008. A GA-based solution approach for balancing printed circuit board assembly lines, *OR Spectrum*, 30, 469-491.

Lusa, A, 2008. A survey of the literature on the multiple or parallel assembly line balancing problem. *European Journal of Industrial Engineering*, 2, 50-72.

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

Martino, L., Pastor, R., 2010. Heuristic procedures for solving the general assembly line balancing problem with setups. *International Journal of Production Research*, 48, 1787-1804.

Miltenburg, J., 2002. Balancing and scheduling mixed-model U-shaped production lines. *International Journal of Flexible Manufacturing Systems*, 14, 1119-151.

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.

Park, K., Park, S., Kim, W., 1997. A heuristic for an assembly line balancing problem with incompatibility, range, and partial precedence constraints. *Computers & Industrial Engineering*, 32, 321-332.

Pastor, R., Andrés, C., Duran, A., Pérez, M., 2002. Tabu search algorithms for an industrial multi-product and multi-objective assembly line balancing problem, with reduction of the tasks dispersion. *Journal of the Operational Research Society*, 53 (12), 1317-1323.

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.

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.

Tazari, S., Müller-Hannemann, M., Weihe, K., 2006. Workload balancing in multi-stage production processes. *Lecture Notes in Computer Science*, 4007, 49-60.

Wee, T.S., Magazine, M.J., 1982. Assembly line balancing as generalized bin packing. *Operations Research Letters*, 1, 56-58.

Appendix

N	*m*	A_0 = 11-15			A_0 = 16-20			A_0 = 21-25		
		Gap%	*MaxGap%*	*Best%*	*Gap%*	*MaxGap%*	*Best%*	*Gap%*	*MaxGap%*	*Best%*
50-200	**5-10**	0.04	0.32	70.00	0.24	0.98	50.00	0.37	1.94	30.00
	11-20	0.31	2.04	60.00	0.62	2.31	50.00	0.49	1.69	40.00
	21-30	0.29	2.20	70.00	0.78	2.82	40.00	2.04	5.71	30.00
	31-40	3.30	28.34	50.00	0.65	3.15	70.00	1.35	6.58	30.00
201-400	**5-10**	0.03	0.14	40.00	0.15	0.77	30.00	0.33	0.67	30.00
	11-20	0.20	0.75	20.00	0.87	1.84	20.00	0.57	1.13	20.00
	21-30	0.76	2.71	40.00	0.51	1.48	30.00	0.59	2.21	30.00
	31-40	1.11	3.54	20.00	0.94	2.66	20.00	0.71	1.74	30.00
401-600	**5-10**	0.01	0.02	60.00	0.08	0.41	40.00	0.12	0.39	10.00
	11-20	0.08	0.41	50.00	0.35	1.32	30.00	0.35	1.04	20.00
	21-30	0.58	1.51	10.00	0.48	1.51	30.00	0.60	1.51	10.00
	31-40	0.68	2.67	20.00	0.78	2.41	30.00	1.18	2.29	0.00
601-800	**5-10**	0.02	0.04	10.00	0.06	0.47	10.00	0.07	0.19	10.00
	11-20	0.05	0.23	30.00	0.27	0.92	0.00	0.41	1.68	40.00
	21-30	0.12	0.30	20.00	0.77	2.62	10.00	0.95	2.48	10.00
	31-40	0.60	1.72	0.00	0.48	1.86	30.00	0.53	1.51	30.00
801-1000	**5-10**	0.02	0.03	0.00	0.03	0.07	10.00	0.09	0.39	10.00
	11-20	0.05	0.11	20.00	0.10	0.35	20.00	0.28	1.05	20.00
	21-30	0.22	0.69	20.00	0.36	0.75	10.00	0.53	1.58	20.00
	31-40	0.13	0.37	30.00	0.52	2.61	40.00	0.45	0.88	20.00
N	*m*	A_0 = 26-30			A_0 = 31-35			A_0 = 36-40		
		Gap%	*MaxGap%*	*Best%*	*Gap%*	*MaxGap%*	*Best%*	*Gap%*	*MaxGap%*	*Best%*
50-200	**5-10**	0.22	0.62	40.00	0.29	1.10	60.00	0.35	1.82	50.00
	11-20	0.28	1.13	50.00	0.38	1.84	70.00	1.20	5.03	50.00
	21-30	0.81	5.28	70.00	0.56	3.54	60.00	0.44	2.84	80.00
	31-40	2.22	9.18	50.00	0.61	3.07	70.00	2.88	18.18	70.00
201-400	**5-10**	0.28	1.16	40.00	0.41	0.92	20.00	0.35	0.94	20.00
	11-20	0.53	1.83	10.00	0.53	1.71	40.00	0.71	2.62	30.00
	21-30	0.94	2.62	20.00	0.74	2.69	30.00	0.83	4.60	20.00
	31-40	1.68	3.73	20.00	1.22	3.57	30.00	1.69	5.01	20.00
401-600	**5-10**	0.43	0.97	10.00	0.42	1.39	10.00	0.34	1.45	40.00
	11-20	0.26	0.92	30.00	0.40	1.18	40.00	0.56	1.55	10.00
	21-30	0.64	1.38	0.00	0.96	3.18	10.00	0.71	2.17	20.00
	31-40	0.81	3.68	40.00	0.67	1.78	10.00	1.14	2.15	10.00
601-800	**5-10**	0.14	0.46	30.00	0.25	0.76	20.00	0.19	0.65	30.00
	11-20	0.36	1.26	20.00	0.43	1.21	30.00	0.55	1.43	20.00
	21-30	0.40	1.28	40.00	0.53	2.33	20.00	0.52	1.67	20.00
	31-40	0.84	2.03	20.00	0.36	1.44	60.00	0.54	1.26	30.00
801-1000	**5-10**	0.12	0.29	20.00	0.22	0.47	20.00	0.17	0.40	10.00
	11-20	0.50	1.71	20.00	0.18	0.50	40.00	0.45	0.97	0.00
	21-30	0.35	1.08	20.00	0.43	0.83	10.00	0.21	0.70	50.00
	31-40	0.77	2.76	10.00	0.61	3.08	40.00	0.52	2.02	10.00

Table 6. Detailed results of the procedure *SA_A2* for for the low versatile instances.

The values shown in Tables 6 correspond to the average values obtained with the procedure *SA_A2* for the 10 instances with low versatile tasks in each data range. The same results are reported in Table 7 for the instances with high versatile tasks.

N	*m*	A_0 = 11-15			A_0 = 16-20			A_0 = 21-25		
		Gap%	*MaxGap%*	*Best%*	*Gap%*	*MaxGap%*	*Best%*	*Gap%*	*MaxGap%*	*Best%*
50-200	**5-10**	0.31	0.89	30.00	0.37	2.10	30.00	0.26	1.17	30.00
	11-20	0.69	2.59	40.00	0.69	3.39	20.00	1.21	3.60	40.00
	21-30	0.89	4.67	30.00	1.71	4.58	20.00	0.69	4.14	50.00
	31-40	1.77	9.92	50.00	0.84	5.42	50.00	1.53	5.00	50.00
201-400	**5-10**	0.01	0.05	50.00	0.02	0.11	60.00	0.31	1.16	30.00
	11-20	0.17	0.95	40.00	0.32	1.48	30.00	0.56	2.27	10.00
	21-30	0.58	2.17	30.00	1.48	2.94	10.00	0.89	2.01	30.00
	31-40	1.54	3.31	10.00	1.57	3.68	30.00	0.97	3.55	10.00
401-600	**5-10**	0.01	0.04	40.00	0.14	0.57	30.00	0.22	0.91	20.00
	11-20	0.05	0.18	40.00	0.47	1.62	20.00	0.31	0.98	20.00
	21-30	0.30	1.17	30.00	0.98	2.08	10.00	1.35	3.30	0.00
	31-40	0.42	0.88	20.00	0.91	2.36	20.00	1.12	3.49	20.00
601-800	**5-10**	0.01	0.02	50.00	0.03	0.22	40.00	0.21	0.49	10.00
	11-20	0.12	0.49	30.00	0.37	1.28	10.00	0.39	1.08	40.00
	21-30	0.28	1.23	20.00	0.34	1.42	20.00	0.78	1.77	0.00
	31-40	0.57	1.67	10.00	0.65	2.57	30.00	0.72	1.73	10.00
801-1000	**5-10**	0.01	0.03	30.00	0.01	0.02	50.00	0.05	0.16	30.00
	11-20	0.11	0.32	10.00	0.12	0.52	20.00	0.36	0.98	20.00
	21-30	0.14	0.34	10.00	0.23	0.90	40.00	0.53	1.56	10.00
	31-40	0.25	0.61	20.00	0.44	1.07	20.00	0.86	2.50	20.00

N	*m*	A_0 = 26-30			A_0 = 31-35			A_0 = 36-40		
		Gap%	*MaxGap%*	*Best%*	*Gap%*	*MaxGap%*	*Best%*	*Gap%*	*MaxGap%*	*Best%*
50-200	**5-10**	0.57	1.74	30.00	0.29	0.74	30.00	0.91	3.27	20.00
	11-20	0.46	1.29	40.00	0.60	1.64	20.00	1.00	4.56	30.00
	21-30	2.32	5.76	30.00	1.10	4.76	40.00	0.35	1.90	60.00
	31-40	2.34	7.59	30.00	1.49	8.47	60.00	0.86	4.98	40.00
201-400	**5-10**	0.41	1.05	0.00	0.40	1.01	20.00	0.31	1.19	20.00
	11-20	0.17	0.62	50.00	0.37	0.82	10.00	0.91	2.50	0.00
	21-30	0.73	1.97	40.00	0.75	3.06	20.00	0.54	1.78	40.00
	31-40	1.46	3.14	10.00	0.95	3.38	40.00	1.75	4.00	10.00
401-600	**5-10**	0.54	1.07	10.00	0.27	0.60	10.00	0.28	0.98	30.00
	11-20	0.57	1.23	0.00	0.39	1.10	30.00	0.49	1.45	20.00
	21-30	0.55	1.52	20.00	0.82	2.77	30.00	0.72	2.36	30.00
	31-40	0.56	0.92	30.00	0.90	3.94	40.00	0.65	3.44	50.00
601-800	**5-10**	0.11	0.45	30.00	0.25	1.11	10.00	0.39	1.61	0.00
	11-20	0.65	1.62	10.00	0.26	0.75	30.00	0.38	1.01	10.00
	21-30	0.53	2.26	30.00	0.42	0.92	10.00	0.43	1.63	30.00
	31-40	0.51	1.42	30.00	0.57	1.47	30.00	0.75	2.06	20.00
801-1000	**5-10**	0.17	0.80	10.00	0.21	0.93	10.00	0.12	0.29	20.00
	11-20	0.40	0.90	0.00	0.37	0.78	10.00	0.22	0.60	30.00
	21-30	0.48	2.63	50.00	0.59	1.61	0.00	0.48	0.91	10.00
	31-40	0.30	0.96	20.00	0.50	2.96	50.00	0.37	1.01	30.00

Table 7. Detailed results of the procedure *SA_A2* for the high versatile instances.