Mathematical models and heuristic methods for the assembly line balancing problem with hierarchical worker assignment

Nícolas P. Campana\textsuperscript{a}, Manuel Iori\textsuperscript{b} and Mayron César O. Moreira\textsuperscript{a}

\textsuperscript{a}Departamento de Ciência da Computação, Universidade Federal de Lavras, Lavras, Brazil;\textsuperscript{b}Department of Sciences and Methods for Engineering, University of Modena and Reggio Emilia, Reggio Emilia, Italy

ABSTRACT

This paper proposes new algorithms for the assembly line balancing problem with hierarchical worker assignment (ALBHW). The ALBHW appears in real industrial contexts, where companies deal with a multi-skilled workforce. It considers task execution times that vary depending on the worker type to whom the task is assigned. Qualification levels among workers are ranked hierarchically, where a lower qualified worker costs less but requires larger execution times then a higher qualified one. The aim is to assign workers and tasks to the stations of an assembly line, in such a way that cycle time and precedence constraints are satisfied, and the total cost is minimised. In this paper, we first present a mathematical model and improve it with preprocessing techniques. Then, we propose a constructive heuristic and a variable neighbourhood descent that are useful to solve large instances. Extensive computational experiments on benchmark instances prove the effectiveness of the algorithms.

ARTICLE HISTORY

Received 23 April 2020
Accepted 24 December 2020

KEYWORDS

Assembly lines; hierarchical worker assignment; mathematical model; variable neighbourhood descent; heuristics

1. Introduction

Assembly lines are essential parts of manufacturing systems for large-scale production (see, e.g. Becker and Scholl 2006, Cortés, Oniev a, and Guadix 2010, Otto and Otto 2014 and Dolgui et al., “Scheduling in Production, Supply Chain and Industry 4.0 Systems by Optimal Control: Fundamentals, Stateof-the-Art and Applications,” 2019). An assembling process consists of a fixed set of tasks (indivisible elements), which need to be executed on stations while taking into account technological precedence relations. The simple assembly line balancing problem (SALBP) is the basic problem encountered when optimising assembly systems. It considers straight assembly lines and aims at assigning tasks to stations in such a way that task precedence constraints are satisfied, and the total amount of time for each station does not exceed a maximum productive rate called cycle time. Two main SALBP variants have been considered in the literature: the SALBP-1 aims at minimising the number of stations for a given cycle time, whereas the SALBP-2 aims at minimising the cycle time for a fixed number of stations. We refer the reader interested in the SALBP literature to Baybars (1986), Scholl and Becker (2006), Becker and Scholl (2006), Boysen, Fliedner, and Scholl (2007), and Battaïa and Dolgui (2013), among others.

In this paper, we study the assembly line balancing problem with hierarchical worker assignment (ALBHW). The problem has been introduced by Sungur and Yavuz (2015) and generalises the SALBP-1. It considers task processing times that depend on the workers, and workers that are divided into hierarchical skill levels. Workers need to be assigned to a station that contains tasks that are compatible with their skills. A worker with higher qualifications incurs a higher cost but executes tasks in shorter times. The goal is to find a minimum-cost assignment of tasks and workers to the stations of the line while satisfying maximum cycle time as well as precedence constraints. The problem is NP-hard in the strong sense. As stressed in Sungur and Yavuz (2015), applications of the ALBHW are numerous and can be found in contexts such as health-care, military, manufacturing, and maintenance systems. In addition, studying hierarchical worker assignment is of interest not only for the ALBHW but also because this characteristic might be found in many other types of assembling and disassembling problems (see, e.g. Ozceylan et al. 2019), just to cite some.

Our contribution to the solution of the ALBHW is threefold. First, we propose a new formulation that strengthens the mathematical model by Sungur and
Yavuz (2015). Second, we implement a station-based constructive heuristic that combines a number of task priority and worker priority selection rules. Third, we present a variable neighbourhood descent (VND) algorithm (Mladenović and Hansen 1997) composed of two neighbourhood procedures, both based on the use of mixed integer linear programs (MILP). The computational tests that we performed on the benchmark problem instances show that the new formulation improves the previous one for what concerns both the optimality gap and the execution time. The improvement is sometimes small but consistent on all attempted instances. Instances with up to 50 tasks can be solved to proven optimality in a matter of seconds, whereas for large-sized instances, involving 100 tasks, we need to content us with heuristic solutions. The VND obtained the best overall performance, producing an average improvement of 1% from the best-known solution values and an average optimality gap of 6.35%, requiring short computing times.

The remainder of the paper is organised as follows. In Section 2, we present the formal definition of the ALBHW. Section 3 surveys the related literature, focusing on multi-skilled and heterogeneous worker assignments, and on labour-cost variants. Section 4 presents the mathematical model by Sungur and Yavuz (2015), as well as our improvements. The heuristic procedures that we implemented are described in Section 5, and the computational experiments are discussed in Section 6. Final remarks and conclusions are given in Section 7.

2. Formal problem definition

We consider a single product straight assembly line. Let $T = \{1, 2, \ldots, n\}$ be the set of indivisible tasks to be allocated in the line, and $S = \{1, 2, \ldots, m\}$ be the set of stations, where $m$ is an upper bound on the number of stations required to process all tasks (e.g., $m = n$). We use the notation $i \preceq j$ to state that task $i$ must be executed before task $j$ in the line, i.e., the index of the station processing $i$ should be lower than or equal to the index of the station processing $j$. The task precedence network is represented by a topological ordered direct graph $G = (T, E)$, where $E = \{(i, j) : i < j\}$. Figure 1 gives an example (inspired from a benchmark instance by Sungur and Yavuz 2015) of a precedence graph involving 13 tasks and 10 precedences.

Let $K = H = \{1, 2, \ldots, l\}$ be two coincident sets denoting, respectively, the set of task types and that of workers types, where $l$ stands for the number of qualification levels. We use index $k$ to denote a task type, and index $h$ to denote a worker type. In this notation, the workers of type 1 are the most qualified, while those of type $l$ are the least qualified. Each task $i \in T$ has a type $k_i \in K$, and can only be executed by workers of type $h$ satisfying $h \leq k_i$.

Each task $i$ is also characterised by an execution time $t_{ih}$ that depends from the worker type $h$ that executes $i$. We assume that the execution time of each task increases when lower-qualified workers are employed, that is, $t_{i1} \leq t_{i2} \leq \cdots \leq t_{il}$. The right top part of Figure 1 gives an example of execution times. If a task requires a minimum qualification level higher than the one possessed by

![Figure 1.](attachment:image) (colour online) Precedence graph (left) and processing times (right) for an ALBHW instance.
a worker, then the worker cannot execute such task. In our notation, this is simply represented by an extremely large value in the time matrix.

The cost of assigning a worker of type $h$ to a station is defined by $c_{h}$, for $h \in H$ and is proportional to its qualification level, so that $c_1 \geq c_2 \geq \ldots \geq c_l$. Given a cycle time $C$, that is, a maximum total execution time allowed at each station, the ALBHW is to assign workers and tasks to stations in order to minimise the total cost, while satisfying constraints on qualifications levels, cycle time, and task precedences. In Figure 2, we present an optimal solution for the instance given in Figure 1. We use $h(s)$ to denote the worker type assigned to station $s$, and $\bar{C}(s)$ to denote the total execution time computed according to the worker/task assignment in the station. In this example, we have $C = 600$, $c_1 = 100$, $c_2 = 70$, and $c_3 = 49$, and so the solution cost is 319.

3. Brief literature review

The interest of the scientific community in incorporating heterogeneity in assembly-line problems has increased over the years. Surveying the entire literature produced in the field is out of the scope of our work. Nevertheless, in this section, we provide a brief literature review that focuses on those aspects that are most relevant to the problem we tackle. In particular, we focus on articles that take into account (i) manufacturing systems with skilled level workers, (ii) heterogeneity over task execution times for each worker, and (iii) hierarchical labour-cost assembly lines. We refer the reader interested in the vast area of assembly line systems to the surveys by Becker and Scholl (2006), Cortés, Onieva, and Guadix (2010), Battaïa and Dolgui (2013), and Otto and Otto (2014). Generalisations of the SALBP involving different task precedence constraints can be found in Dell’Amico, Díaz, and Iori (2012) and Kramer, Dell’Amico, and Iori (2017), among others. These last works also gave inspiration to the improvement techniques that we present below in Sections 4.1 and 4.2. Mansoor (1968) introduced worker performance skills to define task execution times in assembly line systems. He also proposed a constructive heuristic based on positional weighting criteria. Bartholdi III and Eisenstein (1996) considered a case study at Toyota Sewn Products Management System, where workers were classified according to different speed levels. Their results show that sequencing workers from the slowest to the fastest can produce a reasonable line balance. Gel, Hopp, and Van Oyen (2002) considered the possibility of sharing tasks among workers whose task execution times differ by a multiplicative factor. Their study uses a Markov decision process to find good-quality solutions. This approach was later extended by Hopp, Tekin, and Van Oyen (2004), who included cross-training schemes during task assignments. Techawiboonwong, Yenradee, and Das (2006) presented a mathematical model that considers the presence of both permanent (skilled) and temporary (unskilled) workers, as well as specific constraints on the worker assignment on some stations. In the same context, Corominas, Pastor, and Plans (2008) applied a model to solve a rebalancing problem arising in motorcycle assembly lines. They considered given incompatibilities among groups of tasks and aimed at minimising the number of temporary workers employed. In their experimental tests, optimal solutions were found for some realistic instances within a limited computational effort. Koltai, Tatay, and Kalló (2014) presented a similar approach, but focused, instead, on workforce skills in the context of bicycle assembly lines.

Studies considering paced lines and workforce assignments have also attracted the attention of the scientific community. Battaia et al. (2015) studied a paced line with products of different types, where a sequence of products is given, and task assignments and sequencing are known for all stations. Workers are identically skilled, and their assignment to stations can be changed dynamically depending on the current state of the line. The objective is to minimise the number of workers. The authors proposed a model and deterministic and randomised heuristics. Computer experiments proved the efficiency of the proposed approaches. Dolgui et al. (2018) studied the possibility of having parallel stations at the line design. The problem they addressed considers an upper bound on the number of workers assigned to an operation, precedence relationships between tasks, and cycle time constraints. The processing time of an operation is dependent on the number of workers assigned to it. The objective is to minimise the number of workers allocated. The authors provided proof of NP-hardness, a mathematical model, a set of heuristics, and a series of computational experiments over real
and simulated data. They also described problem similarities with the multi-mode project scheduling and multiprocessor scheduling. Delorme et al. (2019) studied a problem that appears as part of a complex industrial environment. They considered paced mixed-lines, with fully skilled workers, with the objective of minimising the number of workers allocated. Besides a new mathematical formulation, they introduced an NP-hardness proof, a dynamic programming algorithm, and the description of polynomial and pseudo-polynomial algorithms for special cases. Lian et al. (2018) consider a bi-objective approach for a worker assignment problem in seru production systems. These assembly lines have a new type of work-cell-based manufacturing system, where most of the tasks are manual, assisted by hand tools, or lightweight equipment. Workers differ from their skills and proficiency levels. They study proposed a mathematical model, minimising the deviation workload from workers and the serus installed at the line. They also propose an adaptation of the NSGA-II algorithm to solve 100 medium-sized instances. According to the tests, heterogeneous workers with diversified competency perform well in balancing inter-seru workload. Liu and Liu (2019) also included workforce satisfaction in their solution methods. They proposed a mathematical model and an ε-constraint method. The authors reported a Pareto front obtained with a small-sized example with 10 tasks, 6 parallel machines, and 3 workers.

We also mention studies concerning multi-skilled workers at U-shaped lines (Nakade and Nishiwaki 2008), two-sided assembly lines (Tapkan, Özbakır, and Baykasoğlu 2016), project scheduling problems (Karam, Attia, and Duquenne 2017), hybrid flow shop environments (Liu and Yang 2019), automotive contexts (Martignago, Battàia, and Battini 2017; Yang, Lee, and Kim 2020), and even problems solved as a generalised assignment (Moussavi, Mahdjoub, and Grunder 2018). We refer to De Bruecker et al. (2015) and Dolgui et al., “Workforce Planning and Assignment in Mixed-Model Assembly Lines as a Factor of Line Reconfigurability: State of the Art,” (2019) for a survey on workforce planning under different skills. Miralles et al. (2007) introduced the assembly line worker assignment and balancing problem (ALWABP), as a problem arising in the context of sheltered worker centres for disabled (SWD) in Valencia (Spain). In the ALWABP, task execution times vary according to the workers, not all workers can perform all tasks, and the aim is to minimise the required cycle time. The authors solved the problem by utilising a mathematical model. Since the ALWABP is NP-hard, heuristic methods were also developed. Among these, we mention clustering search by Chaves, Miralles, and Lorena (2007), beam search by Blum and Miralles (2011), tabu search by Moreira and Costa (2009), genetic algorithms by Moreira et al. (2012), Mutlu, Polat, and Supciller (2013) and Moreira and Ritt (2019), and variable neighbourhood search by Polat et al. (2016). Concerning exact methods for the ALWABP, we highlight the branch-and-bound approaches by Miralles et al. (2008), Borba and Ritt (2014), and Vila and Pereira (2014).

Concerning recent extensions of the ALWABP, Oksuz, Buyukozkan, and Satoglu (2017) proposed a mathematical model, a genetic algorithm, and an artificial bee colony algorithm for the U-shaped ALWABP type-E. Computational tests were performed with three data sets. The authors indicated as further research avenues the study of new algorithms and of a mixed-model variant of the problem. Janardhanan, Li, and Nielsen (2019) addressed the two-sided ALWABP. They proposed a mathematical model that effectively solves (through CPLEX) small-sized instances with less than 16 tasks, and a migrating birds optimisation (MBO) algorithm. The MBO computationally outperformed other metaheuristics as partial swarm optimisation, genetic algorithm, artificial bee colony algorithms, and simulated annealing. The particular ALWABP case in which the heterogeneity derives from the presence of robots is known as the robotic assembly line balancing problem. Cost-oriented approaches, which minimise the robot selection and task allocation costs, have been proposed by Rubinovitz, Bukchin, and Lenz (1993), Bukchin and Tzur (2000), and Pereira, Ritt, and Vásquez (2018).

Hierarchical workforce scheduling (HWS) is a class of problems that has, as its basis, the objective to find the most economical mix of workers that satisfies given labour requirements and worker characteristics. Since the seminal work by Emmons and Burns (1991), several authors have attempted to solve HWS problems. Hung and Emmons (1993) proposed a mathematical model for an HWS variant with compressed workweek arrangements and multiple shifts (rotation and downward worker substitutability). Hung (1994) minimised the labour costs in an HSW problem through a heuristic method, considering the possibility of a single-shift off-day. Still taking a single-shift off-day in seven-day-a-week industries, Narasimhan (1997) implemented an algorithm that minimises the labour costs subject to a pre-defined demand of workers and desired work characteristics. Seçkiner, Gökçen, and Kurt (2007) and Pastor and Corominas (2010) extended the approach by Bilionnet (1999), who studied an HWS case with variable demand, by including compressed workdays and suitability of task assignments. More recently, Ozguven and Sungur (2013) proposed five mathematical models to solve an HWS variant with divisible and indivisible tasks, by minimising total costs.
As far as we know, Sungur and Yavuz (2015) is the only work directly focused on the ALBHW. Apart from introducing the problem, the authors also presented a mathematical model and performed extensive tests on a set of instances adapted from Otto, Otto, and Scholl (2013). Using CPLEX 12.3, their formulation found optimal solutions for small-sized instances with 20 tasks but could not prove optimality for several medium-sized instances involving 50 tasks. These results motivate our work on the development of improved models and new effective heuristics for the problem.

4. Mathematical models

We first introduce the MILP model by Sungur and Yavuz (2015) and then discuss our improvements. We follow the notation given in Section 2. In addition, we use $P_i \subseteq T$ to denote the set of direct predecessors of task $i$, $i \in T$, and $F_i \subseteq T$ to denote the set of direct successors of task $i$, $i \in T$. We also compute the transitive closures of these sets, and use $P_i^*$ and $F_i^*$ to denote, respectively, all (direct and indirect) predecessors and successors of a task $i$, $i \in T$. Consider for example Figure 1, where we have $F_2 = \{5, 7, 8, 9\}$ and $F_2^* = \{5, 7, 8, 9, 11, 12, 13\}$.

The MILP model that we developed is based on the four families of decision variables given in Table 1, three of which are binary and one integer.

The model by Sungur and Yavuz (2015), called for short MSY (Model Sungur Yavuz), is as follows:

$$\text{(MSY)} \quad \min \sum_{h \in H} c_h z_h \tag{1}$$

subject to:

$$\sum_{h \in H, h \leq k_i} x_{ihs} = 1 \quad \forall i \in T \tag{2}$$

$$\sum_{h \in H} y_{hs} = v_s \quad \forall s \in S \tag{3}$$

$$\sum_{i \in T} x_{ihs} \leq n y_{hs} \quad \forall h \in H, s \in S \tag{4}$$

$$\sum_{s \in S} y_{hs} = z_h \quad \forall h \in H \tag{5}$$

Table 1. Decision variables.

| $v_s \in \{0, 1\}$ | takes the value 1 if station $s \in S$ is opened; |
| $x_{ihs} \in \{0, 1\}$ | takes the value 1 if a worker of type $h$ performs task $i$ in station $s$; |
| $y_{hs} \in \{0, 1\}$ | takes the value 1 if a worker of type $h$ is assigned to station $s$; |
| $z_h \in \mathbb{Z}^+$ | gives the total number of workers of type $h$ assigned to the stations. |

The objective function (1) minimizes the total cost of the assembly line. Constraints (2) guarantee that each task is assigned to a station. Constraints (3) impose the assignment of exactly one worker to a station if such station is opened. Coupling constraints between $x_{ihs}$ and $y_{hs}$ variables are ensured by (4). Constraints (5) determine the total amount of workers of type $h$ that are assigned to the stations. Constraints (6) establish the precedence relations between the tasks. Constraints (7) prevent the total workload of each opened station to exceed the cycle time. Inequalities (8) allow a station $s + 1$ to be opened only if the previous station $s$ is also opened. Constraints (9)-(12) define the domains of the variables. This model has $O(|T||H||S|)$ variables and $O(|T|^2 + |H||S|)$ constraints.

4.1. Preprocessing

We used two preprocessing techniques, both aimed at strengthening the mathematical formulation. In the first technique, we calculate the lower bound on the number of stations that separate every two tasks in $E$. We then create a new graph and insert arcs having precedence weights equal to the computed lower bound values. The second technique uses the lower bound values computed by the first technique to calculate the earliest and latest possible station for each task in the graph. Recall, indeed, that $G$ does not have precedence weights, so, if $(i,j) \in E$, then $i$ and $j$ can be assigned to the same station. The inclusion of new arcs and precedence weights can be beneficial for creating stronger cuts and valid inequalities (to be described next in Section 4.2). Both our two preprocessing techniques share this scope, but the first technique builds upon the cycle time restriction, whereas the second uses the precedence.

The first technique creates an auxiliary graph $G'$, starting from the original graph $G$, as follows. Let $G'_{ij} = (N'_{ij}, A'_{ij})$ be a subgraph of $G'$, where $N'_{ij}$ denotes the subset of tasks in all paths connecting two tasks $i,j \in N$ having $F_i^* \cap P_j^* \neq \emptyset$, i.e. $N'_{ij} = (F_i^* \cap P_j^*) \cup \{i,j\}$. In addition, let
\(A'_{ij}\) denote the subset of arcs that belong to all paths from \(N'_{ij}\). Let \(LB_{ij}\) be a lower bound on the number of stations required to process all tasks in \(G'_{ij}\). We compute \(LB_{ij} = \left\lfloor \sum_{k \in N'_{ij}} t_{kj}/C \right\rfloor - 1\) as the continuous lower bound on the processing times of the tasks, by considering the case in which they are executed by the fastest worker (of type 1). We then associate each \(LB_{ij}\) value with its corresponding arc \(A_{ij}\). If \(F^+_i \cap P^+_i = \emptyset\), then we set \(LB_{ij} = 0\). These values are used to produce improved inequalities and are the starting point of the next preprocessing technique. Let \(Z_{vnd}\) be the cost of the solution found by the VND procedure. We define \(m_{ub} = |Z_{vnd}/c_i|\), which is an upper bound on the number of stations.

In the second technique, we solve the longest path problem using the classical critical path method (CPM) to obtain the earliest and latest station index to which any task \(i \in T\) can be assigned, denoted by \(e_i\) and \(l_i\) respectively. We create a new auxiliary graph \(\bar{G} = (\bar{N}, \bar{A})\), based on the \(LB_{ij}\) values. The arc values of \(\bar{G}\) are first copied into \(\bar{G}\). Let \(\bar{N} = N' \cup \{0, n+1\}\), where 0 and \(n+1\) are two dummy tasks that represent the beginning and the end of all activities, respectively. To create the new arcs in the graph, we first update \(\bar{A} = A' \cup \{(0, i)\}, \forall i \in T\) having \(P_i = \emptyset\), thus imposing 0 as source vertex. We then update \(\bar{A} = \bar{A} \cup \{(j, n+1)\}, \forall j \in T\) having \(P_j = \emptyset\), thus imposing \(n+1\) as sink. After this update, we have a graph in which all vertices are connected directly or indirectly with the source and the sink. With the first step of the CPM, we are able to define the earliest station \(e_i\) in which any task \(i\) can be processed. With the second step, we compute the latest station \(l_i\) in which \(i\) can be processed. In detail, the value of \(l_i\) is obtained by first setting \(l_{n+1} = m_{ub}\). Then, we take the transpose graph of \(\bar{G}\), named \(\bar{G}'\), and compute the earliest station of its tasks, \(\bar{e}_i\), by the CPM. The output of the preprocessing procedure is an interval \([e_i, l_i]\) of stations, \(\forall i \in T\).

Figure 3 shows the new graph \(\bar{G}\) created by the preprocessing procedures. The dashed arcs were added by the CPM procedure. Each pair of values near a node gives the indices of the earliest and latest stations, respectively, for the node. The red arc is the lower bound number of stations between tasks 2 and 11, given by \(LB_{2,11} = 1\) and \(LB_{2,12} = LB_{2,13} = 0\). Note that we omitted all the arcs that produce \(LB_{ij} = 0\) to simplify the example.

### 4.2. Improvements to the mathematical model

We developed some techniques for improving the linear relaxation of the MSY model and removing useless variables. First, we set the number of stations in the model to \(m = m_{ub}\). Then, we apply two reformulations of constraints (4) and (7). We start by strengthening \(n\) in constraints (4) by a sufficient large constant \(M_h \leq n\), thus obtaining:

\[
\sum_{i \in T} \alpha_i t_{ih} \leq M_h y_{hs} \quad \forall h \in H, s \in S \tag{13}
\]

The value of \(M_h\) is computed by considering the maximum number of tasks that can be assigned to a station, while satisfying the cycle time \(C\) without precedence constraints and assuming an assignment of only \(h\)-type workers to the stations. Such value is obtained, for each \(h \in H\), by solving the MILP problem:

\[
M_h = \max \sum_{i \in T : h \leq k_i} \alpha_i \tag{14}
\]

subject to:

\[
\sum_{i \in T : h \leq k_i} t_{ih} \alpha_i \leq C \tag{15}
\]

\[
\alpha_i + \alpha_j \leq 1 \quad \forall i, j \in T : LB_{ij} > 0 \tag{16}
\]

\[
\alpha_i \in \{0, 1\} \quad \forall i \in T : h \leq k_i \tag{17}
\]

where \(\alpha_i\) is a binary variable that takes the value 1 if task \(i\) is in the subset of maximum cardinality, and 0 otherwise. Model (14)–(17) is a maximum cardinality knapsack problem with additional incompatibilities imposed by (16). The maximum cardinality knapsack problem is easy, but the addition of the incompatibilities makes it an NP-hard problem. Nevertheless, the above MILP model can be solved very quickly for instances of medium size, as the ones that we face in our tests.

We use a similar technique for constraints (7), replacing the original \(t_{ih}\) values with new ones, \(t'_{ih} \geq t_{ih}\), resulting in:

\[
\sum_{i \in T} \sum_{h \in H, h \leq k_i} t'_{ih} \alpha_i \leq C_v \quad \forall s \in S \tag{18}
\]

Each new value is obtained, for \(i \in T\) and \(h \in H\), by solving a subset sum problem (SSP): given a station of capacity \(C - t_{ih}\) and a set of tasks \(T \setminus \{i\}\), determine the subset of tasks not exceeding the capacity but having largest total time. The SSP solution value, denoted by \(\Delta_{ih}\), is obtained by solving the following model:

\[
\Delta_{ih} = \max \sum_{j \in T \setminus \{i\} : h \leq k_j} t_{jh} \alpha_j \tag{19}
\]

subject to:

\[
\sum_{j \in T \setminus \{i\} : h \leq k_j} t_{jh} \alpha_j \leq C - t_{ih} \tag{20}
\]

\[
\alpha_i + \alpha_j \leq 1 \quad \forall i, j \in T : LB_{ij} > 0 \tag{21}
\]
\[ \alpha_j \in \{0, 1\} \quad \forall j \in T \setminus \{i\} : h \leq k_j \quad (22) \]

where again \( \alpha_j \) takes the value 1 if task \( j \) is selected, and 0 otherwise. The SSP is known to be NP-hard, but, model (19)–(22) too can be quickly solved for instances of medium size. Once the problem is solved, we set \( t_{ih}^f = C - \Delta_{ih} \).

We also modify precedence constraints (6), including the \( LB_{ij} \) values:

\[
\sum_{h \in H, h \leq k_j} \sum_{s \in S} s x_{ihs} + LB_{ij} \\
\leq \sum_{h \in H, h \leq k_j} \sum_{s \in S} s x_{ihs} \quad \forall j \in T, i \in P_j \quad (23)
\]

Furthermore, for each station \( s \) that has the index smaller than the earliest possible or greater than the latest possible, we include the following reduction:

\[ x_{ihs} = 0 \quad \forall i \in T, h \in H, s \leq e_i - 1 \text{ or } s \geq l_i + 1 \quad (24) \]

Let us consider the number \( z_1 \) of workers of type 1 (i.e. those for which \( k_i = 1 \)), and computing a continuos lower bound on their processing times. We consequently add to the model the constraint:

\[ z_1 \geq \left\lceil \sum_{i \in T, k_i = 1} t_{i1} / C \right\rceil \quad (25) \]

The resulting MILP model, called MCIM (from Model Campana Iori Moreira) in the following, is to minimise (1) subject to (2), (3), (5), (8)–(13), (18) and (23)–(25). Our idea, confirmed by the computational results in Section 6.2, is to show how we can improve the existing MSY model and obtain an updated model that has better bounds and optimality gaps. Since our aim is to improve the current model and not create a new one using a different concept and structure, it is expected that the two models adopt different parameter configurations and additional inequalities in the preprocessing step.

### 5. Heuristic algorithms

In this section, we present a constructive heuristic based on the use of different priority rules, and then improve
it by employing a VND algorithm. Our idea is to design fast yet accurate algorithms to solve the ALBHW. Even though we mainly focus on efficiency, computational results (Section 6) show that the procedures obtained good-quality solutions given the time limit imposed.

5.1. Constructive heuristic

The constructive heuristic that we built is inspired by Scholl and Voß (1997) and Moreira et al. (2012) and aims at filling one station at a time. For each station, and for each candidate worker type \( h \in H \), the heuristic creates a set of tasks by using an input task priority rule while respecting precedence and cycle time constraints. Then, a worker type and the corresponding set of tasks just created are selected to be assigned to the station according to a given worker priority rule. The process is repeated until all tasks have been assigned.

We implemented the following task priority rules:

\[
\begin{align*}
\text{(t.1) } & \text{ max successors: select a task } i \text{ having largest } |F_i| \text{ value;} \\
\text{(t.2) } & \text{ max direct successors: select a task } i \text{ having largest } |F_i| \text{ value;} \\
\text{(t.3) } & \text{ largest min processing time: select a task } i \text{ with largest } t_{i1} \text{ value;} \\
\text{(t.4) } & \text{ largest max processing time: select a task } i \text{ with largest } t_{ik} \text{ value;} \\
\text{(t.5) } & \text{ largest min positional weight: select a task } i \text{ with largest } t_{i1} + \sum_{j \in F_i} t_{j1} \text{ value;} \\
\text{(t.6) } & \text{ largest max positional weight: select a task } i \text{ with largest } t_{ik} + \sum_{j \in F_i} t_{jk} \text{ value;} \\
\text{(t.7) } & \text{ largest positional weight: select a task } i \text{ with largest } t_{ih} + \sum_{j \in F_i} t_{jh} \text{ value;} \\
\text{(t.8) } & \text{ shortest min processing time: select a task } i \text{ with shortest } t_{i1} \text{ value;} \\
\text{(t.9) } & \text{ largest direct successors processing time: select a task } i \text{ with largest } |F_i|/(t_{ih} + \sum_{j \in F_i} t_{jh}) \text{ value;} \\
\text{(t.10) } & \text{ largest successors processing time: select a task } i \text{ with largest } |F_i|/t_{ih} \text{ value;} \\
\text{(t.11) } & \text{ largest processing time: select a task } i \text{ with largest } t_{ih} \text{ value;} \\
\text{(t.12) } & \text{ largest worker-type processing time: select a task } i \text{ with largest } t_{ik}, \text{ with } k_i = h; \\
\text{(t.13) } & \text{ shortest worker-type processing time: select a task } i \text{ with shortest } t_{ik}, \text{ with } k_i = h;
\end{align*}
\]

where for positional weight we simply mean the processing time of the task and of all its successors. Other techniques that we attempted did not produce improvements in the computational performance, and were thus disregarded.

In addition, we implemented the following worker priority rules:

\[
\begin{align*}
\text{(w.1) } & \text{ look ahead worker: select a worker of type } h, \text{ by estimating the total cost in case } h \text{ would not be chosen anymore. We group the tasks not assigned yet, named } T, \text{ and allocate them to their worker type } k_i \text{ (the one with highest cost that performs task } i). \text{ Let } \lambda_h \text{ be the approximated solution cost in case no further worker of type } h \text{ would be chosen. Then, taking } T_{h'} = \{ i \in T; h' = k_i \}, \text{ we define } \lambda_h = \sum_{h' \neq h} c_{h'} |T_{h'}/C|. \text{ Let } c_{curr} \text{ be the cost of the partial solution under construction and recall } c_{h} \text{ is the cost of worker type } h. \text{ We select the worker type } h \text{ that produces the smallest } \{c_{curr} + c_{h} + \lambda_h\} \text{ value.} \\
\text{(w.2) } & \text{ smallest cost per task assigned: select a worker type } h \text{ that has the smallest } c_{h}/|T_h| \text{ value, where } T_h \text{ is the set of tasks assigned to worker type } h; \\
\text{(w.3) } & \text{ smallest cost per task processing time assigned: select a worker type } h \text{ that has the smallest } c_{h}/(\sum_{i \in T_h} t_{ih}) \text{ value; } \\
\text{(w.4) } & \text{ largest time per station: select a worker type } h \text{ that has the largest } \sum_{i \in T_h} t_{ih} \text{ value.}
\end{align*}
\]

The pseudocode of the constructive heuristic is given in Algorithm 1. It receives in input the set \( T \) of tasks, the worker priority rule, \( w \_rule \), and the task priority rule, \( t \_rule \). Starting from an empty station, each step of the main loop operates as follows. Line 7 defines the set \( \tilde{T} \), composed by candidate tasks that respect the precedence constraints at the current station. Line 9 creates a subset \( T_h \subseteq \tilde{T} \), for each worker type \( h \), by respecting the cycle time restriction and using the rule \( t \_rule \). If the worker of type \( h \) cannot execute a task chosen by \( t \_rule \), then a new task is chosen with the largest time for the worker \( h \), always considering the precedence or cycle time constraints. The worker priority rule \( w \_rule \) is used at Line 11 to select the best worker according to its cost and its task set \( T_h \). Lines 12–14 perform the necessary updates on the data structures. The algorithm finally returns the set \( W \) of selected workers, and the tasks assigned to the stations \( S \).

5.2. Improving heuristic solutions

We developed a VND procedure that makes use of two MILP-based neighbourhood procedures. The studies of James and Almada-Lobo (2011) and Moreira, Miralles, and Costa (2015) inspired the idea to embed MILP solvers’ resources in our local search procedures. According to Talbi (2009), the VND heuristic iterates neighbourhood structures successively until it reaches a local
optimum. Briefly, consider a neighbour structure \( N_k \), an initial solution \( \text{sol} \), and a set of neighbours of \( \text{sol} \) denoted by \( N_k(\text{sol}) \). If through \( N_k(\text{sol}) \) there is no possibility to find a better solution than \( \text{sol} \), then we replace \( N_k \) with \( N_{k+1} \). Every time we find a better solution than the current one, we return to the first neighbourhood and restart the search. This strategy is more effective when neighbourhoods are complementsaries since one neighbour’s local optimum is not the local optimal of the other neighbour. In our case, we adopt two neighbourhood structures: the first one performs task relocations (Section 5.2.1), and the second one allows worker and task reassignments (Section 5.2.2).

Throughout the procedure, the upper bound on the number of stations is set to the number of stations produced by the solution \( \text{sol}_\text{CH} \) found by the constructive heuristic of Section 5.1. Algorithm 2 shows the pseudocode of the VND. A function \( f(z) \) that maps a solution \( z \) to an objective function value is used to evaluate solutions. The initial solution \( \text{sol}_\text{CH} \) is copied into the incumbent \( \text{sol}_\text{best} \), which is then modified iteratively by means of \( k_{\max} = 2 \) neighbourhood procedures \( N_k \). Such procedures are described in detail in the next two sections.

### 5.2.1. Task relocations

The first local search procedure, called MILP-1, allows tasks to be reassigned to stations that are adjacent to the station in which they were assigned to in the input solution. Let \( \tilde{m} \) be the number of stations in the input solution. Suppose task \( i \) was assigned to station \( s_i \), and let \( \mathcal{S}_i \) be the set of stations to which \( i \) can be reassigned. Formally, we set: \( \mathcal{S}_i = \{1, 2\} \) if \( s_i = 1; \mathcal{S}_i = \{\tilde{m} - 1, \tilde{m}\} \) if \( s_i = \tilde{m} \); and \( \mathcal{S}_i = \{s_{i-1}, s_i, s_{i+1}\} \) if \( 2 \leq s_i \leq \tilde{m} - 1 \). Procedure MILP-1 finds the reallocations of tasks that is optimal with respect to these limited sets by solving the following model:

\[
\min \sum_{h \in H} c_h \sum_{s=1}^{\tilde{m}} y_{hs}
\]

subject to (3)–(12) and:

\[
\sum_{h \in H, s \in \mathcal{S}_i} x_{ih} = 1 \quad \forall i \in T
\]

In practice, this is equivalent to the original model above in Section 4, but its size is consistently reduced by replacing constraints (2) by (27).

### 5.2.2. Worker and task reallocation

The second local search procedure, MILP-2, allows the tasks to be relocated in any position (while respecting the constraints) but requires that the numbers of workers for each type remain unchanged with respect to the input solution. To this aim, we create auxiliary binary variables \( \beta_s \), for \( s = \{1, 2, \ldots, \tilde{m}\} \), taking the value 1 if and only if

---

**Algorithm 1** ALBHW constructive heuristic

```plaintext
1: procedure CONSTRUCTIVE_HEURISTIC(\( T, w\_rule, t\_rule \))
2: \( \mathcal{S}_s \leftarrow \emptyset, s = 1, \ldots, m; \)
3: \( \mathcal{W}_s \leftarrow \emptyset, s = 1, \ldots, m; \)
4: \( s = 0; \)
5: while \( T \neq \emptyset \) do
6: \( s = s + 1; \)
7: \( \tilde{T} \leftarrow \text{AvailableTasks}(T); \)
8: for each worker type \( h \) do
9: \( T_h \leftarrow \text{TaskAssignment}(\tilde{T}, t\_rule, h); \)
10: end for
11: \( h^* \leftarrow \text{WorkerAssignment}(T, w\_rule); \)
12: \( \mathcal{S}_s \leftarrow T_{h^*}; \)
13: \( \mathcal{W}_s \leftarrow h^*; \)
14: \( T = T \setminus T_{h^*}; \)
15: end while
16: return \( \text{sol}_\text{CH} = \{\mathcal{W}_1, \ldots, \mathcal{W}_s, \mathcal{S}_1, \ldots, \mathcal{S}_s\} \);
17: end procedure
```

---

**Algorithm 2** VND algorithm

```plaintext
1: procedure VND(\( \text{sol}_\text{CH}, k_{\max} \))
2: \( k \leftarrow 1; \)
3: \( \text{sol}_\text{best} \leftarrow \text{sol}_\text{CH} \)
4: while \( k \leq k_{\max} \) do
5: \( \text{sol}_\text{curr} \leftarrow N_k(\text{sol}_\text{best}); \)
6: if \( f(\text{sol}_\text{curr}) \leq f(\text{sol}_\text{best}) \) then
7: \( \text{sol}_\text{best} \leftarrow \text{sol}_\text{curr} \)
8: \( k \leftarrow 1 \)
9: else
10: \( k \leftarrow k + 1 \)
11: end if
12: end while
13: return \( \text{sol}_\text{best} \)
14: end procedure
```

---
station $s$ has no task assigned to it. We then let $\alpha_s \in \mathbb{R}$, for $s = \{1, 2, \ldots, \tilde{m}\}$, be the cost of the worker assigned to $s$ when $\beta_s = 1$. Let $\tilde{z}_h$ be the number of workers of type $h$ selected in the input solution. We obtain the following model:

$$\max \sum_{s=1}^{\tilde{m}} \alpha_s$$

subject to (3)–(11) and:

$$\sum_{i=1}^{\tilde{m}} \gamma_{hs} = \tilde{z}_h \quad \forall h \in H$$

$$1 - \beta_s \leq \sum_{i \in T} \sum_{h \in H, h \leq k_i} x_{ihs} \quad s = 1, \ldots, \tilde{m}$$

$$\sum_{i \in T} \sum_{h \in H, h \leq k_i} x_{ihs} \leq |T|(1 - \beta_s) \quad s = 1, \ldots, \tilde{m}$$

$$\alpha_s \leq \sum_{h \in H} c_h \gamma_{hs} \quad s = 1, \ldots, \tilde{m}$$

$$\alpha_s \leq c_1 \beta_s \quad s = 1, \ldots, \tilde{m}$$

$$\beta_s \in \{0, 1\} \quad s = 1, \ldots, \tilde{m}$$

$$\alpha_s \in \mathbb{R} \quad s = 1, \ldots, \tilde{m}$$

The objective function (28) maximises the savings obtained by excluding a worker from the assembly line. Inequalities (30) and (31) are the coupling constraints between variables $\beta$ and $x$, whereas (32) and (33) relate, respectively, $\alpha$ with $y$ and with $\beta$.

### 5.3. A numerical example

In this section, we demonstrate how the heuristic procedures perform. We use the same instance of Section 1, with $w_1 = 1.1$, $w_2 = 0.7$ and $C = 600$. Moreover, we choose the combination of worker and task rules for the configuration $(w_1, w_2)$ denoted by $w.3$ and $t.12$, respectively, as an example. First, we describe the behaviour of the constructive heuristic (Algorithm 1). We list each step below.

- We start with $\bar{T} = \{1, 2, 3\}$ as the available tasks, based on the precedence graph (Line 7).
- In Line 9, the algorithm chooses a set of tasks iteratively, based on the task priority rule. In detail, for each worker, the algorithm selects a subset of tasks respecting the precedence constraints and the maximum cycle time. Since the procedure is iterative, when a task $i$ is selected, we remove the precedence arcs, allowing other tasks to be chosen in the next steps.
- Let $T_1 = \{2, 1, 3, 10\}$, $T_2 = \emptyset$, and $T_3 = \emptyset$ be the results of the loop of Lines 8-10 in the heuristic.
- The only possible option is to select $h^* = 1$ (Line 11).
- After selecting task 2, the tasks $\{7, 8, 5, 9\}$ become available, but they are not considered because all these tasks are of type 2. The same happens when task 3 is selected, unlocking tasks $\{4, 6, 10\}$. But note that the last chosen task is task 10, which is of type 2 (Line 7).
- If the task rule cannot select any task in the available tasks using the $t$ rule, the procedure picks the task with largest processing time with the worker $h$ (at line 9) respecting the constraints (Line 8).
- In the second loop, we have $T_1 = \{4, 5, 8\}$, $T_2 = \{5, 9, 7\}$, and $T_3 = \emptyset$ (Lines 8-10).
- Let $c'_h$ be the calculated value in line 11. We obtain the values $c'_1 = (100/596) = 0.17$, $c'_2 = (70/587) = 0.12$. Using the $w.3$ priority rule, we select $h^* = 2$ (Line 11).
- This procedure continues until all the tasks have been assigned to the stations (Line 5).
- The output from the algorithm is $S_1 = \{2, 3, 1, 10\}$, $S_2 = \{5, 9, 7\}$, $S_3 = \{6, 8, 11\}$, $S_4 = \{13, 12\}$, $S_5 = \{4\}$, and $W_1 = \{1\}$, $W_2 = \{2\}$, $W_3 = \{2\}$, $W_4 = \{3\}$, $W_5 = \{1\}$ (Line 16).

In the VND procedure (Algorithm 2), we start with neighbourhood $N_1$, which is the task reallocation MILP (Line 5). We also list the next steps below, as we did for Algorithm 1.

- The MILP improves the objective value rearranging the tasks (Line 5) as $S_1 = \{1, 2, 3, 7\}$, $S_2 = \{8, 9, 10\}$, $S_3 = \{5, 12, 13\}$, $S_4 = \{4, 6, 11\}$ and $W_1 = \{1\}$, $W_2 = \{2\}$, $W_3 = \{2\}$, $W_4 = \{1\}$, decreasing the objective value from 389 to 340 (Lines 6-7).
- Since we have an improvement, we continue with $N_1$, where it is again possible to obtain an improvement (Line 8).
- The neighbourhood counters $k$ increases when we have no improvement, proceeding to the next neighbourhood $N_2$, which is the worker and task reallocation MILP (Line 10).
- We impose that the number of workers of each type is the same from the input $\tilde{z} = \{2, 1, 1\}$, but the model can choose to assign them in any station. Note that in $N_2$ (Line 5), we solve the same original problem but with a reduced solution space. The model can find the optimal solution if (i) the number of $\tilde{z}_h \geq z^*_h, \forall h \in H$, which means that the number of workers type $h$ need to be at least equal to the number of workers type $h$ of the optimal solution, and (ii) if there is enough time to solve it.
- At this point, there are no more possible improvements and the procedure ends with the objective function value of 319 (Line 13). Note that the value 319 is indeed optimal.
6. Computational experiments

In our computational experiments, we evaluate the results obtained by the methods presented in this paper. The evaluation of the mathematical models is given in Section 6.2 and that of the heuristics in Section 6.3. The algorithms have been coded in C++, using Gurobi v9.0 as MILP solver. The tests have been executed on a PC Ubuntu 16.04.6 LTS 64-bit with Intel(R) Xeon(R) CPU E3-1245 v5 3.50GHz and 32GB RAM. The maximum execution time was set to 2 hours for the two models of Section 4, and to 15 s for each execution of the MILP-based neighbourhoods of Section 5.2.

The tests were done using the small-sized and medium-sized instances by Sungur and Yavuz (2015), as well as new large-sized instances that we created to obtain a better evaluation of our methods, as described in Section 6.1. In total, we solved 900 instances. Because of the large size of the testbed that we adopted, we only present aggregate results. All the instances that we used, together with the detailed computational results that we obtained on every single instance, have been made publicly available at https://github.com/NicolasCampana/ALBHW-Instances.

6.1. Benchmark set

To the best of our knowledge, the only benchmark instances for the ALBHW have been proposed by Sungur and Yavuz (2015). They chose 45 small-sized \((n = 20)\) and 45 medium-sized \((n = 50)\) instances among those created by Otto, Otto, and Scholl (2013) for the SALBP-1. For each such instance, they generated 5 ALBHW instances by attempting all combinations of two parameters, \(w_1 \in \{1.10, 1.20\}\) and \(w_2 \in \{0.70, 0.85\}\) and including an additional instance in which both \(w_1\) and \(w_2\) were set to 1. In this way, they created \(45 \times 5 = 225\) small-sized and 225 medium-sized instances. Parameter \(w_1\) is the time factor that defines the time scaling from a worker type to the next, used in such a way that \(t_{i,h+1} = w_1 t_{i,h}\). Parameter \(w_2\) is used instead to determine the cost scaling among the worker types, in such a way that \(c_{h+1} = w_2 c_h\).

To obtain a better evaluation of our methods, we also created a new set of 225 large-sized ALBHW instances having \(n = 100\) tasks and 225 very large-sized instances having \(n = 250\). These have been obtained following the same procedure adopted by Sungur and Yavuz (2015). We thus adopted a random process to choose the SALBP-1 instances from Otto, Otto, and Scholl (2013) with mixed precedence graphs, balancing the amount of ‘extremely tricky’, ‘very tricky’, ‘less tricky’ and ‘tricky’ instances. Additional information about the instances can be found in Sungur and Yavuz (2015) and in our public repository.

6.2. Evaluation of the mathematical models

Table 2 presents the results obtained by the two models, MSY by Sungur and Yavuz (2015) and the newly proposed MCIM. Both models do not have any initial solution as an input. For each model, we give the number of proven optimal solutions \(#\text{opt}\), the number of instances for which at least one feasible solution has been found \(#\text{feas}\), the percentage optimality gaps at the root node \(%\text{gap}_{\text{root}}\) and at the end of the execution \(%\text{gap}\), the number of explored nodes \(#\text{nodes}\) and the total computing time in seconds \(\text{time(s)}\). Each line in the table gives aggregate total (for \#opt and \#feas) or average (for all other columns) values for 45 instances sharing the same value of \(n\), \(w_1\) and \(w_2\). The lines called ‘average/total’ give aggregate values for the 225 instances having the same value of \(n\), and the last ‘overall’ line presents aggregate results for the entire set of 900 instances. The best \%gap values are highlighted in bold.

Both models were very effective in solving the small-sized instances with just 20 items. Some remarkable differences can be noticed, however, already for the cases with \(n = 50\): MSY can only find 148 optima, with an average effort of 2821 seconds an average gap of 1.24%, whereas MCIM obtains 14 more optima, with just 2330 seconds of average effort and producing an optimality gap that is just 0.54%. The difference becomes more evident for the group having \(n = 100\): MSY finds only 33 proven optima and 147 feasible solutions, with an average gap of about 44%, whereas MCIM obtains nine more optima, 24 more feasible solutions, and a smaller gap of 28.89% on average. When \(n = 250\), MCIM found less feasible solutions than MCY, but still more ones. It also obtained better gaps and larger numbers of explored nodes. In any case, it is evident from the results that heuristic algorithms should be developed to tackle the very large-sized instances. Overall, MCIM is more accurate than MSY on all groups of instances. This is achieved at the expense of a slightly larger number of explored nodes, but with a reduction in the computing time. The good MCIM performance can also be imputed to the fact that it has a better root node gap than MSY, especially on the small-sized and medium-sized instances.

Overall, we can conclude that the two models are very effective in solving the small-sized instances but show important drawbacks for the larger-sized instances, for which it is important to invoke heuristic algorithms. We believe it is also important to notice the impact of the hierarchical worker assignment on the difficulty of the instances. This can be observed by comparing the instances having \(w_1 = 1.00\) and \(w_2 = 1.00\), which correspond to the pure SALBP-1 because all workers are equal one another both in terms of cost and of processing times,
6.3. Evaluation of the heuristic algorithms

In Table 3, we present an aggregate evaluation of our heuristic procedures. To facilitate comparison, for MCIM, we resume from Table 2 the number of feasible solutions found (#feas) and the average gap from the lower bound (%gap). For the constructive heuristic, called CH for short in the following, we provide the average solution value (UB) and again the average percentage gap from the lower bound. For the VND, we provide the same information given for CH, but we also report the average improvement with respect to CH, computed as

\[
\text{impr} = \frac{(UB_{VND} - UB_{CH})}{UB_{VND} \times 100},
\]

and the average execution time in seconds (time(s)). We execute the VND with an input coming from the best combination task/worker priority rules of the CH, as a greedy strategy to start the method with a good-quality solution, since it is very fast to run all combinations of priority rules. In any case, if one had to choose a single rule that performs well under various conditions, we would suggest rule w.3 and t.3, due to its performance shown in Table 4. The constructive heuristic is 2.62% away from the lower bound on the instances with 20 tasks, and 3.21% on those with 50 tasks. The VND can improve CH by about 1.41% when \( n = 20 \), achieving a gap equal to 1.42%, and by 1.35% when \( n = 50 \), achieving a gap equal to 2.48%. In these cases, MCIM appears to be a good solution choice. However, as previously noted, MCIM fails in finding feasible solutions for many instances with \( n = 100 \) (namely, 54 instances) and for almost all instances with \( n = 250 \) (namely, 202). For all such cases, both CH and VND quickly found feasible solutions. In addition, they could reduce the gap consistently from the lower bound, from the 28.89% obtained on average by MCIM, to just 8.15% on average for CH.

### Table 3. Comparison between the mathematical models (45 instances per line).

| Instance | MSY (Sungur and Yavuz 2015) | MCIM (new) |
|----------|-----------------------------|-------------|
| \( n \) | \( w_1 \) | \( w_2 \) | #opt | #feas | %gap | %gap_{root} | #nodes | time(s) | #opt | #feas | %gap | %gap_{root} | #nodes | time(s) |
| 20 | 1.00 | 1.00 | 45 | 45 | 0.00 | 6.10 | 11 | 0.1 | 45 | 45 | 0.00 | 1.13 | 21 | 0.03 |
| 1.10 | 0.70 | 1.00 | 45 | 45 | 0.00 | 20.83 | 625 | 24.6 | 45 | 45 | 0.00 | 9.54 | 54 | 2.30 |
| 1.10 | 0.85 | 1.00 | 45 | 45 | 0.00 | 12.97 | 5098 | 18.3 | 45 | 45 | 0.00 | 6.29 | 1208 | 2.21 |
| 1.20 | 0.70 | 1.00 | 45 | 45 | 0.00 | 21.81 | 14350 | 57.7 | 45 | 45 | 0.00 | 9.29 | 1380 | 6.10 |
| 1.20 | 0.85 | 1.00 | 45 | 45 | 0.00 | 12.69 | 2689 | 9.88 | 45 | 45 | 0.00 | 6.15 | 956 | 1.61 |
| average/total | 225 | 225 | 0.00 | 14.88 | 5715 | 22.13 | 225 | 225 | 0.00 | 6.48 | 743 | 2.45 |
| 50 | 1.00 | 1.00 | 43 | 45 | 0.27 | 10.06 | 664419 | 359.61 | 43 | 45 | 0.27 | 6.10 | 605562 | 335.80 |
| 1.10 | 0.70 | 1.00 | 31 | 45 | 1.81 | 27.29 | 31106 | 2959.90 | 33 | 45 | 0.64 | 10.74 | 30400 | 2211.88 |
| 20 | 1.00 | 1.00 | 24 | 45 | 0.95 | 18.60 | 430031 | 3685.35 | 27 | 45 | 0.54 | 7.72 | 826847 | 3109.96 |
| 1.20 | 0.70 | 1.00 | 23 | 45 | 2.24 | 29.73 | 183131 | 3892.03 | 29 | 45 | 0.83 | 12.00 | 344168 | 3174.04 |
| 1.20 | 0.85 | 1.00 | 27 | 45 | 0.94 | 19.82 | 183408 | 3210.78 | 30 | 45 | 0.40 | 9.53 | 108349 | 2820.55 |
| average/total | 148 | 225 | 1.24 | 21.10 | 298419 | 2821.0 | 162 | 225 | 0.54 | 9.23 | 383065 | 2330.45 |
| 100 | 1.00 | 1.00 | 30 | 42 | 7.85 | 23.15 | 504302 | 2777.06 | 30 | 45 | 1.37 | 3.03 | 763736 | 2472.60 |
| 1.10 | 0.70 | 1.00 | 1 | 26 | 60.00 | 78.18 | 4709 | 7141.95 | 6 | 31 | 37.22 | 41.96 | 11066 | 6504.71 |
| 1.10 | 0.85 | 1.00 | 1 | 26 | 47.72 | 71.03 | 30459 | 7103.89 | 4 | 29 | 38.71 | 42.80 | 47984 | 6791.12 |
| 1.20 | 0.70 | 1.00 | 0 | 25 | 63.25 | 80.52 | 4307 | 7200.00 | 1 | 32 | 37.09 | 42.96 | 33912 | 7154.06 |
| 1.20 | 0.85 | 1.00 | 1 | 28 | 44.64 | 70.71 | 37473 | 7186.01 | 1 | 34 | 30.06 | 35.08 | 38859 | 7127.46 |
| average/total | 33 | 147 | 44.69 | 64.72 | 116250 | 6281.0 | 42 | 171 | 28.89 | 33.35 | 179111 | 6009.99 |
| 250 | 1.00 | 1.00 | 6 | 18 | 63.39 | 74.37 | 82349 | 6964.76 | 10 | 19 | 58.31 | 58.37 | 384560 | 5767.60 |
| 1.10 | 0.70 | 0 | 2 | 99.66 | 99.67 | 4 | 7200.00 | 0 | 1 | 98.79 | 98.82 | 538 | 7200.00 |
| 1.10 | 0.85 | 0 | 3 | 99.60 | 99.40 | 2 | 7200.00 | 0 | 0 | 100.00 | 100.00 | 740 | 7200.00 |
| 1.20 | 0.70 | 0 | 1 | 99.89 | 99.89 | 50 | 7200.00 | 0 | 1 | 98.58 | 98.71 | 377 | 7200.00 |
| 1.20 | 0.85 | 0 | 3 | 97.48 | 98.49 | 70 | 7200.00 | 0 | 2 | 96.45 | 96.72 | 836 | 7200.00 |
| average/total | 6 | 27 | 91.97 | 94.36 | 16495 | 7152.97 | 10 | 23 | 90.43 | 90.52 | 160082 | 6913.54 |
| overall | 412 | 624 | 34.47 | 48.77 | 109220 | 4069.60 | 439 | 644 | 29.96 | 34.89 | 160082 | 3814.11 |
Table 3. Evaluation of the heuristic algorithms CH and VND (45 instances per line).

| instance | MCIM | CH | VND |
|----------|------|----|-----|
| n        | #feas | %gap | UB  | %gap | %impr | %gap | time(s) |
| 20       | 45    | 0.00 | 688.89 | 4.82 | 677.78 | −3.19 | 2.39 | 0.01 |
| 1.00     | 1.00  | 6   | 536.11 | 1.81 | 553.00 | −0.54 | 1.30 | 3.04 |
| 1.10     | 0.70  | 45  | 626.53 | 2.57 | 619.84 | −1.39 | 1.29 | 2.26 |
| 1.10     | 0.85  | 45  | 568.27 | 1.20 | 564.96 | −0.63 | 0.61 | 3.74 |
| 1.20     | 0.85  | 45  | 648.27 | 2.70 | 641.93 | −1.29 | 1.50 | 2.52 |
| average/total | 613.61 | 2.62  |
| 50       | 45    | 0.27 | 1657.78 | 5.85 | 1637.78 | −2.41 | 3.72 | 0.12 |
| 1.00     | 1.00  | 6   | 1265.64 | 3.99 | 1254.33 | −1.34 | 2.73 | 12.96 |
| 1.10     | 0.70  | 45  | 1488.64 | 2.41 | 1481.24 | −0.58 | 1.85 | 23.46 |
| 1.10     | 0.85  | 45  | 1365.69 | 3.77 | 1353.27 | −1.28 | 2.56 | 17.31 |
| 1.20     | 0.70  | 45  | 1554.31 | 3.73 | 1543.09 | −1.12 | 1.55 | 31.41 |
| 1.20     | 0.85  | 45  | 1466.41 | 3.21 | 1453.94 | −1.35 | 2.48 | 17.05 |
| average/total | 818.06 | 8.15  |
| 100      | 45    | 1.37 | 3231.11 | 3.88 | 3213.33 | −0.91 | 3.03 | 0.33 |
| 1.00     | 1.00  | 6   | 2502.53 | 10.51 | 2477.69 | −1.14 | 9.50 | 46.07 |
| 1.10     | 0.70  | 45  | 2960.00 | 6.49 | 2935.29 | −0.96 | 5.59 | 71.27 |
| 1.10     | 0.85  | 45  | 2707.87 | 11.92 | 2685.76 | −1.02 | 11.03 | 51.67 |
| 1.20     | 0.70  | 45  | 3088.80 | 7.95 | 3067.31 | −0.83 | 7.19 | 74.54 |
| 1.20     | 0.85  | 45  | 2898.06 | 8.15 | 2875.95 | −0.97 | 7.27 | 48.78 |
| average/total | 312.59 | 14.94 |
| 250      | 19    | 58.31 | 7897.78 | 5.34 | 7886.67 | −0.27 | 5.07 | 4.53 |
| 1.00     | 1.00  | 6   | 6044.73 | 0.36 | 17.31 | 86.40 |
| 1.10     | 0.70  | 45  | 7275.98 | 12.99 | 7243.87 | −0.34 | 12.70 | 91.69 |
| 1.10     | 0.85  | 45  | 6677.64 | 21.37 | 6642.82 | −0.35 | 21.09 | 109.71 |
| 1.20     | 0.70  | 45  | 7615.22 | 15.16 | 7602.00 | −0.12 | 15.06 | 70.18 |
| 1.20     | 0.85  | 45  | 7091.18 | 14.49 | 7084.03 | −0.29 | 14.24 | 72.51 |
| average/total | 3021.82 | 7.25  |
| overall  | 644   | 29.96 | 3021.82 | 7.25 | 3005.34 | −1.00 | 6.35 | 35.16 |

Table 4. Number of times in which a combination of worker and task priority rules led to the best CH solution (best results in bold, 180 instances per group).

| (w_1, w_2) | W     | t.1 | t.2 | t.3 | t.4 | t.5 | t.6 | t.7 | t.8 | t.9 | t.10 | t.11 | t.12 | t.13 | total |
|------------|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|------|------|------|-------|
| (1.00, 1.00) | 63   | 489 | 117 | 17  | 0   | 0   | 0   | 4   | 2   | 0   | 0    | 0    | 0    | 0    | 180   |
| (1.10, 0.70) | 81   | 6   | 17  | 0   | 0   | 1   | 1   | 1   | 2   | 0   | 110  | 0    | 0    | 0    | 44    |
| (1.10, 0.85) | 81   | 6   | 17  | 0   | 0   | 1   | 1   | 1   | 2   | 0   | 110  | 0    | 0    | 0    | 44    |
| (1.20, 0.70) | 81   | 6   | 17  | 0   | 0   | 1   | 1   | 1   | 2   | 0   | 110  | 0    | 0    | 0    | 44    |
| (1.20, 0.85) | 81   | 6   | 17  | 0   | 0   | 1   | 1   | 1   | 2   | 0   | 110  | 0    | 0    | 0    | 44    |

which is a useful tool to visualise different algorithmic performances. To generate the graphics, we use the Julia package called BenchmarkProfiles, available at https://github.com/JuliaSmoothOptimizers/BenchmarkProfiles.jl. We compare four methods presented in this study: the two mathematical models (MSY and MCIM) and the two heuristic procedures (CH and VND), in which CH corresponds to the best solution from all combinations of task/worker priority rules. In short, let $I$ be a set of instances, $A$ a set of algorithms, and $c_{pa}$ the total cost obtained by algorithm $a \in A$ in instance $p \in I$. The performance ratio of algorithm $a \in A$ in instance $p \in I$
is defined by:

\[ r_{pa} = \frac{\zeta_{pa}}{\min\{\zeta_{pa} : a \in A\}} \]

The performance profile \( \rho_a(\tau) \) of an algorithm \( a \in A \) is given by:

\[ \rho_a(\tau) = \frac{|p \in P : r_{pa} \leq \tau|}{|P|}, \tau \geq 1 \]

The value of \( \rho_a(\tau) \) indicates the probability for algorithm \( a \in A \) that a performance ratio \( r_{pa} \) is within a factor \( \tau \) of the best possible ratio. When a set of instances are compared, the algorithm with a larger \( \rho_a(\tau) \) is preferred.

Figures 4–7 present the results for small, medium, large and very large groups of instances, respectively. We only consider instances for which each of the four methods obtained at least a feasible solution. For small-sized instances (Figure 4), we see a similar performance of MSY and MCIM model. If we consider a loss of 10% efficiency, the VND and the CH heuristics achieve almost 100% of the best-known solutions. An identical performance of the four methods is noted when we allow a loss of 30% efficiency. In Figure 5, we observe the dominant behaviour of the MCIM model, which obtained practically all of the best solutions. The model MSY also has a good performance, getting 100% of the benchmark if we take a loss of efficiency greater than 5%. The VND and the CH needs a loss of efficiency of more than 15% to solve all the problems.

In Figures 6 and 7, we note that the VND heuristic starts to outperform the other methods due to the complexity to obtain high-quality solutions found by the models and the simplicity of CH. Concerning solution quality, these analyses confirm a behaviour we expected, that applying heuristic methods (especially the VND) is more appropriate for large and very large instances, while...
the mathematical models provide high-quality solutions for small and medium instances.

In Table 4, we try to obtain some insight into the performance of all task and worker priority rules (see Section 5.1) used by CH. Worker rules are summarised as lines in the table, whereas task rules as columns. Differently from the previous tables, in this case, we opted to aggregate instances in groups \((w_1, w_2)\) having the same values of parameters \(w_1\) and \(w_2\), independently of the value taken by \(n\). As previously noted, the first group (1.00, 1.00) corresponds to the SALBP-1 because all workers are equal both in terms of cost and of processing times. For this group, all worker rules are equivalent, and we thus concentrate only on the task rules. For the other four groups, where hierarchical worker assignment has instead an impact, we show the results obtained by both task and worker priority rules. Each cell in the table gives the number of times in which a pair of rules led CH to find the best solution cost. Total values are provided for each task rule in the last line, and each worker rules in the right-most column. The best values for each group are highlighted in bold. For group (1.00, 1.00), the best task rule is t.3, which consists of selecting the task with the largest processing time using the worker type 1, which is a relaxation to the SALBP-1. For group (1.10, 0.70), the best result is obtained by the combination of rules w.3 and t.12. This corresponds to selecting the tasks with the shortest processing times and the workers that have the shortest time per task assigned. Task rule t.12 is also very effective for the groups (1.1, 0.85) and (1.20, 0.70), both combined with w.3. Task rule t.3 is the best for the last group (1.20, 0.85) and is the one obtaining the highest total number of best results, 296 out of 900. Overall, all rules have a positive impact on the algorithm. For some of them, this is small (as for t.8 and t.11), but for others, it is quite remarkable (as for t.1, t.3, and t.12). There is no clear winner among the worker priority rules, as all of them have a balanced positive impact on the algorithm.
Figure 8. (colour online) Average gaps from MCIM solution costs (only for cases where MCIM found a feasible solution) obtained by the different heuristic procedures, divided by groups \((w_1, w_2)\).

Figure 9. (colour online) Average gaps from MCIM solution costs (only for cases where MCIM found a feasible solution) obtained by the different heuristic procedures, divided by size.

In Figures 8 and 9, we provide a further analysis aiming at evaluating the impact of each heuristic component, selecting only those instances for which MCIM found a feasible solution. In Figure 8, we show the overall gap for each group \((w_1, w_2)\). As the analysis by size is also interesting to show the effectiveness of the procedures, we used Figure 9, organised by each size \(n\) its the overall gaps. We then computed the percentage gap of a certain heuristic configuration, say, \(H\), from the upper bound of the model, as \((UB_H - UB_{MCIM}) / UB_H \times 100\). We tested CH, CH improved by MILP-1 (run once) and CH improved by the VND. We can notice a decrease in the gap that is consistent on all groups and all sizes.

7. Conclusions
We studied the assembly line balancing problem with hierarchical worker assignment (ALBHW), a problem...
that requires to assign tasks and workers to the stations of an assembly line, to minimise costs while satisfying precedence and cycle time constraints. Workers are divided into different qualification levels, and lower qualified workers cost less but also require larger execution times for the tasks. We proposed some improvements to an existing mathematical model, and then developed a constructive heuristic and a variable neighbourhood descent algorithm. The improved mathematical model is quite efficient in solving instances with up to 50 tasks, but for larger instances it is convenient to adopt the heuristic algorithms. Several instances, now made publicly available on the Internet, remain unsolved to proven optimality, and future more powerful exact and heuristic algorithms are envisaged for solving the problem.

The inclusion of the hierarchical worker assignment makes the problem much more difficult to solve. Mathematical models incur, indeed, in much larger root node optimality gaps and, consequently, in much larger execution times. Making use of a multi-skill workforce to its full extent may have a critical impact in several production activities not only assembling ones, but also disassembling and scheduling in general. Interesting future research directions include thus the addition of hierarchical worker assignment characteristics to other classes of problems. The implementation of other neighbourhood-based metaheuristics such as iterated local search and variable neighbourhood search is also a promising future research topic. Concerning the mathematical models, a reasonable idea to feed a feasible solution into MSY and MCIM formulations to guide the search from the given solution, given that Gurobi supports this idea by MIP start. Finally, based on the successful utilisation of branch, bound, and remember algorithm in other assembly line balancing problems (Vila and Pereira 2014; Li, Kucukkoc, and Tang 2020; Li et al. 2020, e.g.), we believe that an adaptation of such an algorithm could be promising for the ALBHW.

Acknowledgements

This research was partially supported by Fundação de Amparo a Pesquisa de Minas Gerais (FAPEMIG), EDITAL PIBIC/FAPEMIG, PRP No 06/2016. The authors thank Banu Sungur and Yasemin Yavuz for kindly supplying the small- and medium-sized instances that they created. Finally, the authors also thank the valuable suggestions given by the four anonymous reviewers and by the Associate Editor.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This research was partially supported by Fundação de Amparo a Pesquisa de Minas Gerais (FAPEMIG), EDITAL PIBIC/FAPEMIG, PRP No 06/2016.

Notes on contributors

Nicolas Porto Campana is an undergraduate student at the Computer Science Department at the Federal University of Lavras, Brazil. In 2017/2018, he was granted a one-year scholarship by the FAPEMIG-Brazil to develop an assembly line balancing project as a junior researcher. He was awarded the best undergraduate work in the 50th Brazilian Symposium of Operations Research. His interests fields are combinatorial optimisation, integer programming, and graph theory.

Manuel Iori is an associate professor of Operations Research at the University of Modena and Reggio Emilia. He has published more than 80 papers in international journals in the areas of combinatorial optimisation and logistics, with a focus on exact and heuristic algorithms for travelling salesman, vehicle routing, bin packing, production and scheduling problems. He is member of the research centres CIRRELT, INDAM, AIRI and INTERMECH MORE. He collaborates with companies for the solutions of real-world optimisation problems, by designing and developing solution methods and decision support systems.

Mayron César O. Moreira is a professor at the Department of Computer Science of the Federal University of Lavras, Minas Gerais, Brazil. He has a Ph.D. in Computer Science and Computational Mathematics at the University of São Paulo, with a research stage at Polytechnic University of Valencia (Valencia, Spain) and CIRRELT (Montréal, Canada). In 2019, he was a visiting professor at the University of Modena and Reggio Emilia (Reggio Emilia, Italy). His experience concerns combinatorial optimisation problems, especially scheduling problems, assembly line balancing problems, vehicle routing problems, irregular cutting and packing problems, and operations research applied in agriculture. He also collaborates with companies for the design of efficient and scalable solutions to real-world problems.

References

Bartholdi III, John J., and Donald D. Eisenstein. 1996. “A Production Line that Balances Itself.” Operations Research 44 (1): 21–34.

Battaïa, Olga, Xavier Delorme, Alexandre Dolgui, Johannes Hagemann, Anika Horlemann, Sergey Kovalev, and Sergey Malyutin. 2015. “Workforce Minimization for a Mixed-model Assembly Line in the Automotive Industry.” International Journal of Production Economics 170: 489–500.

Battaïa, Olga, and Alexandre Dolgui. 2013. “A Taxonomy of Line Balancing Problems and Their Solution Approaches.”
Baybars, Ilker. 1986. “A Survey of Exact Algorithms for the Simple Assembly Line Balancing Problem.” *Management Science* 32 (8): 909–932.

Becker, Christian, and Armin Scholl. 2006. “A Survey on Problems and Methods in Generalized Assembly Line Balancing.” *European Journal of Operational Research* 168 (3): 694–715.

Billonnet, Alain. 1999. “Integer Programming to Schedule a Hierarchical Workforce with Variable Demands.” *European Journal of Operational Research* 114 (1): 105–114.

Blum, Christian, and Cristobal Morillas. 2011. “On Solving the Assembly Line Worker Assignment and Balancing Problem Via Beam Search.” *Computers & Operations Research* 38 (2): 328–339.

Boysen, Nils, Malte Fliedner, and Armin Scholl. 2007. “A Classification of Assembly Line Balancing Problems.” *European Journal of Operational Research* 183 (2): 674–693.

Bukchin, Joseph, and Michal Tzur. 2000. “Design of Flexible Assembly Line to Minimize Equipment Cost.” *IIE Transactions* 32 (7): 585–598.

Chaves, Antonio A., Cristobal Morillas, and Luiz A. N. Lorena. 2007, October. “Clustering Search Approach for the Assembly Line Worker Assignment and Balancing Problem.” In *International Conference on Computers and Industrial Engineering*, 1469–1478.

Corominas, Albert, Rafael Pastor, and Joan Plans. 2008. “Balancing Assembly Line with Skilled and Unskilled Workers.” *Omega* 36 (6): 1126–1132.

Cortés, Pablo, Luis Onieva, and José Guadix. 2010. “Optimising and Simulating the Assembly Line Balancing Problem in a Motorcycle Manufacturing Company: A Case Study.” *International Journal of Production Research* 48 (12): 3637–3656.

De Bruecker, Philippe, Jorne Van den Bergh, Jeroen Beliën, and Erik Demeulemeester. 2015. “Workforce Planning Incorporating Skills: State of the Art.” *European Journal of Operational Research* 243 (1): 1–16.

Dell’Amico, Mauro, José C. Díaz Díaz, and Manuel Iori. 2012. “The Bin Packing Problem with Precedence Constraints.” *Operations Research* 60 (6): 1491–1504.

Delorme, Xavier, Alexandre Dolgui, Sergey Kovalev, and Mikhail Y. Kovalyov. 2019. “Minimizing the Number of Workers in a Paced Mixed-model Assembly Line.” *European Journal of Operational Research* 272 (1): 188–194.

Dolan, Elizabeth D., and Jorge J. Moré. 2002. “Benchmarking Optimization Software with Performance Profiles.” *Mathematical Programming* 91 (2): 201–213.

Dolgui, Alexandre, S. Ehsan Hashemi Petroodi, Sergey Kovalev, Mikhail Y. Kovalyov, and Simon Thevenin. 2019. “Workforce Planning and Assignment in Mixed-Model Assembly Lines As a Factor of Line Reconfigurability: State of the Art.” *IFAC-PapersOnLine* 52 (13): 2746–2751.

Dolgui, Alexandre, Dmitry Ivanov, Suresh P. Sethi, and Boris Sokolov. 2019. “Scheduling in Production, Supply Chain and Industry 4.0 Systems by Optimal Control: Fundamentals, State-of-the-Art and Applications.” *International Journal of Production Research* 57 (2): 411–432.

Dolgui, Alexandre, Sergey Kovalev, Mikhail Y Kovalyov, Sergey Malyutin, and Ameer Soukhal. 2018. “Optimal Workforce Assignment to Operations of a Paced Assembly Line.” *European Journal of Operational Research* 264 (1): 200–211.

Emmons, Hamilton, and Richard N. Burns. 1991. “Off-day Scheduling with Hierarchical Worker Categories.” *Operations Research* 39 (3): 484–495.

Gel, Esma S., Wallace J. Hopp, and Mark P. Van Oyen. 2002. “Factors Affecting Opportunity of Worksharing As a Dynamic Line Balancing Mechanism.” *IIE Transactions* 34 (10): 847–863.

Hopp, Wallace J., Eylem Tekin, and Mark P. Oyen. 2004. “Benefits of Skill Chaining in Serial Production Lines with Cross-Trained Workers.” *Management Science* 50 (1): 83–98.

Huang, Rudy. 1994. “Single-shift Off-day Scheduling of a Hierarchical Workforce with Variable Demands.” *European Journal of Operational Research* 78 (1): 49–57.

Huang, Rudy, and Hamilton Emmons. 1993. “Multiple-shift Workforce Scheduling Under the 3-4 Compressed Workweek with a Hierarchical Workforce.” *IIE Transactions* 25 (5): 82–89.

James, Ross J. W., and Bernardo Almada-Lobo. 2011. “Single and Parallel Machine Capacitated Lotsizing and Scheduling: New Iterative MIP-based Neighborhood Search Heuristics.” *Computers & Operations Research* 38 (12): 1816–1825.

Janardhanan, Mukund Nilakantan, Zixiang Li, and Peter Nielsen. 2019. “Model and Migrating Birds Optimization Algorithm for Two-sided Assembly Line Worker Assignment and Balancing Problem.” *Soft Computing* 23 (21): 11263–11276.

Karam, Ahmed, El-Awady Attia, and Philippe Duquenne. 2017. “A MILP Model for An Integrated Project Scheduling and Multi-skilled Workforce Allocation with Exible Working Hours.” *IFAC-PapersOnLine* 50 (1): 13964–13969.

Koltag, Tamás, Viola Tatay, and Noémi Kalló. 2014. “Application of the Results of Simple Assembly Line Balancing Models in Practice: the Case of a Bicycle Manufacturer.” *International Journal of Computer Integrated Manufacturing* 27 (9): 887–898.

Kramer, Raphael, Mauro Dell’Amico, and Manuel Iori. 2017. “A Batching-move Iterated Local Search Algorithm for the Bin Packing Problem with Generalized Precedence Constraints.” *International Journal of Production Research* 55 (21): 6288–6304.

Li, Zixiang, Zeynab Abidin Çil, Süleyman Mete, and Ibrahim Kucukkoc. 2020. “A Fast Branch, Bound and Remember Algorithm for Disassembly Line Balancing Problem.” *International Journal of Production Research* 58 (11): 3220–3234.

Li, Zixiang, Ibrahim Kucukkoc, and Qihua Tang. 2020. “A Comparative Study of Exact Methods for the Simple Assembly Line Balancing Problem.” *Soft Computing* 24: 11459–11475.

Lian, Jie, ChenGuang Liu, WenJuan Li, and Yong Yin. 2018. “A Multi-skilled Worker Assignment Problem in Seru Production Systems Considering the Worker Heterogeneity.” *Computers & Industrial Engineering* 118: 366–382.

Liu, Ming, and Xin Liu. 2019. “Satisfaction-driven Bi-objective Multi-skill Workforce Scheduling Problem.” *IFAC-PapersOnLine* 52 (13): 229–234.
Liu, Ming, and Xuenan Yang. 2019. "Bi-objective Optimization for Scheduling and Multi-skilled Worker Assignments in the Hybrid Flow Shop." IFAC-PapersOnLine 52 (13): 2128–2133.

Mansoor, E. M. 1968. "Assembly Line Balancing: a Heuristic Algorithm for Variable Operator Performance Levels." Journal of Industrial Engineering 19: 618–629.

Martignago, Michele, Olga Battaia, and Daria Battini. 2017. “Workforce Management in Manual Assembly Lines of Large Products: a Case Study.” IFAC-PapersOnLine 50 (1): 6906–6911.

Miralles, Cristóbal, José P. García-Sabater, Carlos Andrés, and Manuel Cardós. 2008. "Branch and Bound Procedures for Solving the Assembly Line Worker Assignment and Balancing Problem: Application to Sheltered Work Centres for Disabled." Discrete Applied Mathematics 156 (3): 352–367.

Miralles, Cristóbal, José P. García-Sabater, Carlos Andrés, and Manuel Cardos. 2007. “Advantages of Assembly Lines in Sheltered Work Centres for Disabled. A Case Study.” International Journal of Production Economics 110 (2): 187–197.

Mladenović, Nenad, and Pierre Hansen. 1997. “Variable Neighborhood Search.” Computers & Operations Research 24 (11): 1097–1100.

Moreira, Mayron C. O., and Alysson M. Costa. 2009. “A Minimalist Yet Efficient Tabu Search Algorithm for Balancing Assembly Lines with Disabled Workers.” Simpósio Brasileiro de Pesquisa Operacional 660–671.

Moreira, Mayron C. O., Cristóbal Miralles, and Alysson M. Costa. 2015. “Model and Heuristics for the Assembly Line Worker Integration and Balancing Problem.” Computers & Operations Research 54: 64–73.

Moreira, João P. G., and Marcus Ritt. 2019. “Evolving Task Priority Rules for Heterogeneous Assembly Line Balancing.” Proceedings of the 2019 IEEE Congress on Evolutionary Computation (CEC), 1–8.

Moreira, Mayron C. O., Marcus Ritt, Alysson M. Costa, and Antonio A. Chaves. 2012. “Simple Heuristics for the Assembly Line Worker Assignment and Balancing Problem.” Journal of Heuristics 18 (3): 505–524.

Moussavi, S. E., M Mahdjoub, and O Grunder. 2018. “A Hybrid Heuristic Algorithm for the Sequencing Generalized Assignment Problem in An Assembly Line.” IFAC-PapersOnLine 51 (2): 695–700.

Mutlu, Özcan, Olcay Polat, and Aliye A. Supciller. 2013. “An Iterative Genetic Algorithm for the Assembly Line Worker Assignment and Balancing Problem of Type-II.” Computers & Operations Research 40 (1): 418–426.

Nakade, Koichi, and Rei Nishiwaki. 2008. “Optimal Allocation of Heterogeneous Workers in a U-shaped Production Line.” Computers & Industrial Engineering 54 (3): 432–440.

Narasimhan, Rangarajan. 1997. “An Algorithm for Single Shift Scheduling of Hierarchical Workforce.” European Journal of Operational Research 96 (1): 113–121.

Oksuz, Mehmet Kursat, Kadir Buyukozkan, and Sule Itir Satoglu. 2017. “U-shaped Assembly Line Worker Assignment and Balancing Problem: A Mathematical Model and Two Meta-heuristics.” Computers & Industrial Engineering 112: 246–263.

Otto, Alena, and Christian Otto. 2014. "How to Design Effective Priority Rules: Example of Simple Assembly Line Balancing." Computers & Industrial Engineering 69: 43–52.

Otto, Alena, Christian Otto, and Armin Scholl. 2013. “Systematic Data Generation and Test Design for Solution Algorithms on the Example of SALBPGen for Assembly Line Balancing.” European Journal of Operational Research 228 (1): 33–45.

Ozceylan, Eren, Can B. Kalyayci, Aknur Gnr, and Surendra M. Gupta. 2019. “Disassembly Line Balancing Problem: A Review of the State of the Art and Future Directions.” International Journal of Production Research 15 (16): 4805–4827.

Ozguven, Cemal, and Banu Sungur. 2013. “Integer Programming Models for Hierarchical Workforce Scheduling Problems Including Excess Off-days and Idle Labour Times.” Applied Mathematical Modelling 37 (22): 9117–9131.

Pastor, Rafael, and Albert Corominas. 2010. “A Bicriteria Integer Programming Model for the Hierarchical Workforce Scheduling Problem.” Journal of Modelling in Management 5 (1): 54–62.

Pereira, Jordi, Marcus Ritt, and Óscar C. Vásquez. 2018. “A Memetic Algorithm for the Cost-oriented Robotic Assembly Line Balancing Problem.” Computers & Operations Research 99: 249–261.

Polat, Olcay, Can B. Kalyayci, Özcan Mutlu, and Surendra M. Gupta. 2016. “A Two-phase Variable Neighbourhood Search Algorithm for Assembly Line Worker Assignment and Balancing Problem Type-II: An Industrial Case Study.” International Journal of Production Research 54 (3): 722–741.

Rubinovitz, Jacob, Joseph Bukchin, and Ehud Lenz. 1993. “RALB–A Heuristic Algorithm for Design and Balancing of Robotic Assembly Lines.” CIRP Annals 42 (1): 497–500.

Scholl, Armin, and Christian Becker. 2006. “State-of-the-art Exact and Heuristic Solution Procedures for Simple Assembly Line Balancing.” European Journal of Operational Research 168 (3): 666–693.

Scholl, Armin, and Stefan Voß. 1997. “Simple Assembly Line Balancing – Heuristic Approaches.” Journal of Heuristics 2 (3): 217–244.

Şekkiner, Serap U., Hadi Gökçen, and Mustafa Kurt. 2007. “An Integer Programming Model for Hierarchical Workforce Scheduling Problem.” European Journal of Operational Research 183 (2): 694–699.

Sungur, Banu, and Yasemin Yavuz. 2015. “Assembly Line Balancing with Hierarchical Worker Assignment.” Journal of Manufacturing Systems 37: 290–298.

Talbi, El-Ghazali. 2009. Metaheuristics: From Design to Implementation. Vol. 74. Hoboken, NJ: John Wiley & Sons.

Tapkam, Pnar, Lale Özbakır, and Adil Baykasoğlu. 2016. “Bee Algorithms for Parallel Twosided Assembly Line Balancing Problem with Walking Times.” Applied Soft Computing 39: 275–291.

Tchawwoonwong, Atthawit, Pisal Yenradee, and Sanchoy K. Das. 2006. “A Master Scheduling Model with Skilled and Unskilled Temporary Workers.” International Journal of Production Economics 103 (2): 798–809.

Vila, Mariona, and Jordi Pereira. 2014. “A Branch-and-bound Algorithm for Assembly Line Worker Assignment and Balancing Problems.” Computers & Operations Research 44: 105–114.

Yang, Hyungjoon, Je-Hun Lee, and Hyun-Jung Kim. 2020. “Workforce Assignment with a Different Skill Level for Automotive Parts Assembly Lines.” In IFIP International Conference on Advances in Production Management Systems, 206–212. Springer.