Integer Based Formulation for the Simple Assembly Line Balancing Problem with Multiple Identical Tasks

Celso Gustavo Stall Sikora[a], Thiago Cantos Lopes[a], Daniel Schibelbain[b], Leandro Magatão[a,*]

[a] *Graduate Program in Electrical and Computer Engineering (CPGEI) Federal University of Technology - Paraná (UTFPR), Curitiba, Brazil, 80230-901*
[b] *Renault do Brasil S.A., São José dos Pinhais, Brazil, 83070-900*

Abstract

Assembly lines, especially those with welding procedures, can present several tasks with the same properties. These tasks can be treated as tasks with replicas, simplifying the problem. A Mixed Integer Linear Programming model is presented for the Simple Assembly Line Balancing Problem with Multiple Identical Tasks (or Repeated Tasks). Integer variables were used to define the number of identical tasks performed in each station. Along with variable reduction rules, the compact formulation presents only a fraction of the variables of equivalent binary models when several repeated tasks are present. Three instances inspired in real assembly lines and adapted benchmark problems with repeated tasks are used to compare the formulations. Using a universal solver, the integer formulation outperformed the binary formulation for the vast majority of instances and achieved competitive results in relation to the efficient procedure SALOME-2 (a dedicated algorithm based on branch-and-bound for Simple Assembly Line Balancing Problem). Grouping identical tasks proved to simplify the problem, allowing the procedure to solve larger instances.

Keywords: Assembly line balancing, Mixed integer linear programming, Identical tasks

Computers & Industrial Engineering, Volume 104, February 2017, Pages 134-144
DOI: 10.1016/j.cie.2016.12.026

1. Introduction

The Assembly Line Balancing Problem (ALBP) is the problem in which a set of product assembly operations is divided among workstations in flow-lines. Firstly defined by Salveson (1955), an ALBP solution is the task partition that maximizes an assembly line efficiency. The allocation of tasks, however, is subjected to technological restrictions in respect of the order tasks are assigned. When a task depends on the conclusion of another operation, a precedence relation is defined.

An ALBP can be described by a graph G in which weighted nodes V are tasks to be allocated in stations S $\{1, ..., M\}$ and edges represent the precedence relations A (Scholl, 1999). An instance

*Corresponding author
Email address: magatao@utfpr.edu.br (Leandro Magatão)

can be defined as $G = \{V, A, t\}$ where V is the set of tasks i $\{1, ..., N\}$, and t_i is the duration time of task i. As a result of precedence relations, P_i (F_i) is defined as the set of direct predecessors (successors) of task i, while A is the task pairs $(i, j) \mid \{i \in V, j \in F_i\}$. The set P_i^* (F_i^*) contains all predecessors (successors) of task i, including direct and indirect relations. As an example, in Fig. 2, Task 1 is a direct predecessor of Task 4 while Task 10 is an indirect successor of Task 1.

Several ALBP variations have been modeled based on characteristics found in the industry. When several products are assembled in the same line, a mixed-model balancing is necessary (Gökcen & Erel, 1998). Further improvement on production levels are possible by integrating the balancing problem with the model sequencing problem (Hamzadayi & Yildiz, 2013; Kucukkoc & Zhang, 2014). Another related problem that can be solved simultaneously is the allocation of workers along the assembly line (Borba & Ritt, 2014; Vilà & Pereira, 2014; Moreira et al., 2015; Ramezanian & Ezzatpanah, 2015; Sikora et al., 2016). Moreover, assembly line may present two operation sides (Lee et al., 2001) or set-up times between operation pairs (Yolmeh & Kianfar, 2012). An effort of systematic classification of models was performed by Boysen et al. (2007) while recent reviews on general topics on ALBP are from Becker & Scholl (2006), Boysen et al. (2008) and Battaïa & Dolgui (2013).

The basic version of the problem, which is the base for most of the its extensions, was labeled by Baybars (1986). In his survey, Baybars defined the Simple Assembly Line Balancing Problem (SALBP) as subjected to several simplification hypothesis (SH):

- (SH-1) All stations are equally equipped and manned
- (SH-2) Task processing time is independent of the workstation
- (SH-3) Any station is able to perform any task
- (SH-4) The assembly line is serial, no feeder or subassembly lines are considered
- (SH-5) The assembly line is designed for the production of a single product
- (SH-6) All problem's parameters are deterministic (processing time and precedence relations)

Baybars (1986) also defines two versions of the SALBP, namely type-1 and type-2. In SALBP-1, the line's cycle time is given and the objective is to minimize the number of workstations needed for the task assignments. On the other hand, SALBP-2 is subjected to a given number of workstations while the cycle time is the optimization focus.

Although extensions of ALBP are a growing trend on the more recent articles, the simple version (SALBP) has been the most intensively studied ALBP in the literature (Battaïa & Dolgui, 2013). The first two integer models for SALBP were proposed by Bowman (1960). Bowman's first model used unnecessary integer variables that were simplified into binary variables by White (1961). The computational capacity at the time was not enough advanced for practical purposes. A decade later, Thangavelu & Shetty (1971) and Patterson & Albracht (1975) developed binary models adapted to be solved with Balas' method (Geoffrion, 1967). Patterson & Albracht (1975) also developed a

reduction approach to the number of variables using the problem structure, resulting in a compact model. Scholl (1999) presents a review on the SALBP models and introduces two SALBP models: one based on task sequencing and other whose assignment is determined by the difference of two binary variables. More recently, Pastor & Ferrer (2009) developed a model containing restrictions that dynamically reduced the search space as function of the incumbent solution. Their model doesn't require an initial upper bound, the restrictions use the intermediary solutions to cut the search field. The several formulations for the precedence relations were discussed by Ritt & Costa (2015), who proposed a model with a tighter precedence restriction.

SALBP formulations are extensively used among further modeling in ALBP extension using mixed-integer solvers (Battaïa & Dolgui, 2013). For SALBP, however, problem specific heuristics, metaheuristics, or exact methods are usually more efficient than solvers. A comparative study between 12 solution methods of different concepts was performed by Pape (2015). Pape compared the performance of genetic algorithm (Falkenauer & Delchambre, 1992; Sabuncuoglu et al., 2000), differential evolutionary algorithm (Nearchou, 2005), ant colony optimization (Bautista & Pereira, 2002), tabu search (Scholl & Voss, 1997; Lapierre et al., 2006), MultiHoffman heuristic (Fleszar & Hindi, 2003), bounded dynamic programming (Bautista & Pereira, 2009), beam search (Blum, 2008), and branch-and-bound (Scholl & Klein, 1997, 1999) for the solution of SALBP-1. The most efficient algorithms are dedicated branch-and-bound or dynamic-programming procedures that use the problem structure to determinate the search procedure.

Scholl & Becker (2006) presented a survey on exact and heuristic procedures highlighting the performance of SALOME-1 (Scholl & Klein, 1997, 1999) and SALOME-2 (Klein & Scholl, 1996) for the SALBP-1 and SALBP-2, respectively. A further development on an exact procedure for SALBP-1 is due to Sewell & Jacobson (2012). Their Branch, Bound, and Remember Algorithm (BB&R) solved to optimality, for the first time, all the 269 SALBP-1 instances of Scholl (1999)'s dataset with an average of only 0.43 seconds. For SALBP-2, however, to the best knowledge of the authors, still no exact algorithm outperforms SALOME-2 developed by Klein & Scholl (1996).

All of the models and procedures presented treat each task individually, assigning a set of variables for each task. Assembly lines with welding procedures can, for example, present several tasks that have similar characteristics. Figure 1, for instance, illustrates 16 tasks divided in three groups (G_1, G_2, and G_3). Within each group, the welding spots are performed in similar positions and have the same function. In an ALBP with identical tasks (repeated tasks with the same processing time and precedence relations), the problem formulation does not benefit from the fact that many tasks can be identical and therefore can be gathered in groups of similar tasks. Here, we propose a representation that benefits from repeated tasks to simplify the problem solving. This way, solution methods are able to compute bigger instances of assembly line balancing problems with these characteristics.

The paper is structured as follows. In Section 2, the definition of identical tasks is proposed, along with the requirement for grouping tasks and their effect on the problem structure. Section 3 presents one review of a binary formulation for SALBP and the new integer variable based

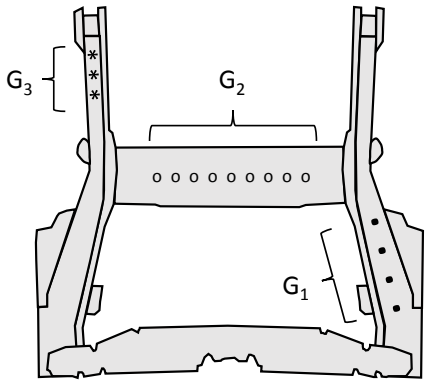


Figure 1: Representation of an automotive part to illustrate a welding procedure. The symbols exemplify welding points. The asterisk and the filled and unfilled circles represent tasks than can be considered identical. Within each group (G_1, G_2, and G_3), the processing time and precedence relations are the same for each task.

formulation modeling groups of tasks. In Section 4 computational experiments are described with the results showing the advantages of an integer formulation. The conclusion remarks are found in Section 5.

2. The simple assembly line with multiple identical tasks problem

Among tasks performed in an assembly line, it is possible that some operations appear as copies of other tasks. Assembling two pieces using screws or welding operations exemplifies such cases: the operation might require multiple screws, welding spots or beads. When procedures have the same duration and the order they are performed is irrelevant, they can be said to be identical tasks.

Figure 2 exemplifies a case in which 15 tasks are constituted of 5 tasks of 3 copies each. Between each group of identical tasks, once they can be performed in any order, there aren't precedence relations. As a result, groups of identical tasks appear as vertical aligned tasks.

In this kind of problem, there might be several answers with identical quality. Suppose the optimal answer requires a workstation to perform Tasks 1 and 2. Another answer could assign Tasks 1 and 3 or 2 and 3 to that workstation, while still resulting in an optimal answer. Once there are identical tasks, equally balanced assignments can be achieved by swapping such tasks.

In one enumerative procedure for SALBP, Jackson (1956) proposed a dominance rule to avoid exploring station loads whose balancing quality is limited by other assignment. Jackson's dominance rule states that Task i potentially dominates Task j if $t_i \geq t_j$, no precedence relations are defined between them and $F_i \supseteq F_j$. If a partial solution of a subproblem includes a Task j in a station, while a dominant Task i is still unassigned, the dominance rule tell us that if we can swap the two tasks without surpassing the cycle time restriction of the problem, the subproblem is dominated and can be left out. In the case of $t_i = t_j$ and $F_i = F_j$, the index of the task is the

tiebreaker: i dominates j if $i < j$.

All identical tasks such as presented in Fig. 2 will fall into the tiebreaker criterion. Once the first of each group of tasks potentially dominates the others, adding precedence relations to force its earlier assignment results in the same optimal balancing. The extra relations produces a more restricted, but equivalent SALBP. The result of this procedure is shown in Fig. 3. Note that in Fig. 2 the groups of tasks are organized vertically. The added precedences in Fig. 3 implies a fixed order in which the tasks must be performed, therefore, they appear horizontally aligned.

A new equivalent integer representation is proposed here. Identical operations can be merged in a task that must be performed multiple times. Figure 4 is the equivalent diagram showed in Fig. 2 with the difference that this representation contains the number of copies each task has in the upper left position of nodes. Even though the problem still has 15 tasks, its representation is simplified when we merge identical tasks. The problem's solution is not only the determination of which tasks are performed in which stations, but how many tasks of a given type are performed in each station.

Any SALBP problem can be described in such way. When no task has a copy, however, the precedence diagram has the same length, but it must also show an additional information: each task appears only once. This approach can only simplify the problem's representation in the presence of repeated tasks. In Section 3, we show an integer model suited to solve problems with the integer representation of Figure 4.

3. Integer Assembly Line Balancing Model

In this section, we present an integer formulation to SALBP that explores the structure of a problem with multiple identical tasks to create a compact model. While most of the models in Section 1 use binary variables to describe whether a task is assigned to a workstation, our formulation is based on integer decision variables. Instead of treating each copy of a task individually, an

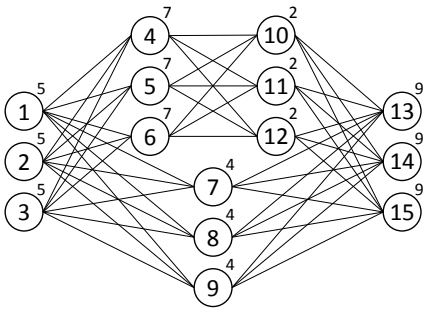


Figure 2: Precedence graph of a SALBP with multiple copies of tasks. No precedence relation between the identical tasks is assumed, so that tasks are represented vertically aligned. The number inside the circle is the index of each task, while their duration time is shown in the upper right position. Tasks are ordered from left to right (pointers are omitted due to the high concentration of arcs in the diagram).

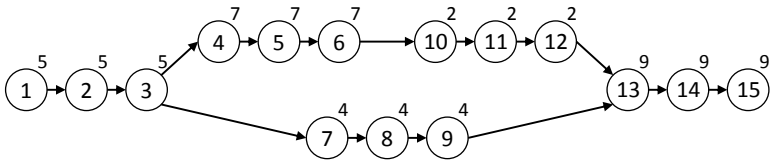


Figure 3: Horizontal aligned equivalent precedence graph of Fig. 2. Precedence relations are used to define an order in which the identical tasks must be performed. These assumed relations result in the horizontal alignment of the tasks in the diagram.

integer variable is used to determine how many copies of a task are assigned to each workstation. The formulations described in this section are used for SALBP-2 instances once the real-world inspired case studies presented in Section 4 have a fixed number of workstations.

In Subsection 3.1, the formulation of Patterson & Albracht (1975) exemplifies the use of binary-based variables. This model is used as a reference to introduce the integer-based model presented in Subsection 3.2. Note that in both binary and integer formulations, due to precedence relations, not every combination of pairs task-station are feasible allocations. For instance, tasks with several followers cannot be allocated in the end of the line. Subsection 3.3 defines how the set FTA (Feasible Task Allocations), firstly described by Patterson & Albracht (1975), is used to eliminate variables that would result in impossible allocations along with the integer formulation modification for the set.

3.1. Binary Approach Model

Although the Patterson & Albracht's original model is described for SALBP-1, due to the conditions of the case study, we describe a SALBP-2 adaptation. The variables are defined as:

$$CT : \text{cycle time}$$

$$x_{ij} : \begin{cases} 1, \text{ if task } i \text{ is assigned to station } j \\ 0, \text{ otherwise} \end{cases} \qquad \forall (i,j) \in FTA$$

The adapted objective function is set to minimize CT. The model is subjected to the following

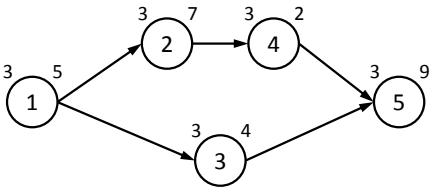


Figure 4: Equivalent precedence graph of Fig. 2 with merged multiple identical tasks. This diagram doesn't represent every task individually, but as groups of tasks. The representation also contains the number of copies each task has in the upper left position.

constraints:

$$\sum_{(i,j)\in FTA} x_{ij} \cdot t_i \leq CT \qquad \forall\ j = 1, ..., M \tag{1}$$

$$\sum_{(i,j)\in FTA} x_{ij} = 1 \qquad \forall\ i = 1, ..., N \tag{2}$$

$$\sum_{(i,k)\in FTA} k \cdot x_{ik} \leq \sum_{(j,k)\in FTA} k \cdot x_{jk} \qquad \forall\ (i,j) \in A \tag{3}$$

$$x_{ij} = \{0, 1\} \qquad \forall\ (i,j) \in FTA \tag{4}$$

In assembly lines, the processing time of each station is the sum of the time spent on each task. The most loaded station is the bottleneck, determining the cycle time of the line (Constraint 1). The balancing is performed assigning all tasks to workstations. The Occurrence Constraint 2 along with the Integrality Constraint 4 assure each task is allocated to a single station. Finally, the precedence relations are implemented through Constraint 3, which forces followers to be allocated not before their predecessors.

Alternative formulations for the precedence relations were proposed in the literature (Bowman, 1960; Thangavelu & Shetty, 1971; Scholl, 1999; Ritt & Costa, 2015). We mention the constraint proposed by Bowman (1960) and improved by White (1961) in Constraint 5. Although this formulation requires more restrictions, they only use 0 or 1 as coefficients. Such values are important for the integer formulation of Subsection 3.2.

$$x_{jl} \leq \sum_{\substack{(i,k)\in FTA \\ k\leq l}} x_{ik} \qquad \forall\ (i,j) \in A,\ (j,l) \in FTA \tag{5}$$

3.2. Integer Approach Model

In this formulation, we define decision variables for each existent type of task. Replicas of a task are treated in a single integer variable. Instead of determining whether a task is assigned to a station, this model's variables decide how many copies of each task are assigned to each station. This way, lines with repeated tasks can be modeled using fewer variables than the binary based model of Section 3.1. The integer formulation uses the following extra parameters to define an instance:

n_i : number of copies of task i

$n_{max_{ij}}$: maximal number of copies of task i that can be allocated in station j

The model variables are defined as:

CT : cycle time

z_{ij} : number of tasks i assigned to station j

$$y_{ij} : \begin{cases} 1, \text{ if all tasks } i \text{ are completed by stations up to } j \\ 0, \text{ otherwise} \end{cases}$$

$(i,j) \in FTA$

While the integer decision variable (z_{ij}) results in the allocation of tasks, y_{ij} is used as a support variable in the precedence relations. This variable indicates whether a task i is completed until station j, i.e., all its copies are assigned at station j or before. The objective function is to minimize CT, subjected to the following constraints:

$$\sum_{(i,j)\in FTA} z_{ij} \cdot t_i \leq CT \qquad \forall\ j = 1, ..., M \tag{6}$$

$$\sum_{(i,j)\in FTA} z_{ij} = n_i \qquad \forall\ i = 1, ..., N \tag{7}$$

$$z_{jk} \leq y_{ik} \cdot n_j \qquad \forall\ (i,j) \in A,\ (i,k) \text{ and } (j,k) \in FTA \tag{8}$$

$$y_{ij} \leq \sum_{\substack{(i,k)\in FTA \\ k\leq j}} \frac{z_{ik}}{n_i} \qquad \forall\ (i,j) \in FTA \tag{9a}$$

$$y_{ij} + n_i - 1 \geq \sum_{\substack{(i,k)\in FTA \\ k\leq j}} z_{ik} \qquad \forall\ (i,j) \in FTA \tag{9b}$$

$$y_{ij} = \{0, 1\} \qquad \forall (i,j) \in FTA \tag{10}$$

$$z_{ij} = \left\{0, 1, ..., n_{max_{ij}}\right\} \qquad \forall (i,j) \in FTA \tag{11}$$

Inequality 6, the cycle time constraint, is identical to the binary Inequality 1. The Occurrence Restriction 7 only differs in terms of the number of copies that should be assigned. While the binary model allocates every task individually, the sum of all assignments in the integer model must be equal to the number of copies. Inequalities 8 and 9 are responsible for the precedence relations. These constraints are derived from Constraint 5: x_{jl} can only assume 1 if any x_{ik} is also 1 for $k \leq l$. For the integer model, however, the assignment condition is that all copies of a precedent task must be assigned before the assignment of a follower task. The right side of Inequality 8 needs y_{ik} as a support variable. According to Constraint 9a, y_{ij} can only assume the value 1, and therefore allow followers to be assigned, if all replicas of z_i are assigned up to station j. Constraint 9b forces y_{ij} to assume 1 in this condition, but this restriction is not necessary for the modeling of precedence relations. Once this extra restriction represented computational gains in some instances, the model used also included Restriction 9b in all the experimental tests. Finally, the Integrality Constraints 10 and 11 forces y_{ij} and z_{ij} to assume discrete values. Along with only allowing an integer number of copies to be assigned to a station, Constraint 11 also presents an upper bound for the number of copies that can be assigned. This upper bound value is further discussed in Subsection 3.3.

3.3. Variable set reduction

For both models, variables are only defined when they are part of the set FTA. This set uses the precedence relations to exclude task assignments that are proven to be unfeasible. FTA is built

based on the earliest and latest station a task can be allocated. For the binary model, according to Patterson & Albracht (1975), the earliest station for a task i can be calculated by adding the time of all tasks that precede i (P_i^*) divided by the cycle time. The ratio measures how many stations at least have to be filled in order to the predecessors of i are all assigned, Equation 12. Once stations' indexes are integer values, the ceiling function is used. Analogously, L_i is calculated based on the duration times of the successors (F_i^*), Equation 13. FTA is defined as all the task-station pairs in which the stations are within the interval between the earliest and latest stations, according to Equation 14. For SALBP-2 instances, an upper bound for the cycle time should be used.

$$E_i = \left\lceil \frac{t_i + \sum_{j \in P_i^*} t_j}{CT} \right\rceil \qquad \forall\ i = 1, ..., N \tag{12}$$

$$L_i = M - 1 + \left\lceil \frac{t_i + \sum_{j \in F_i^*} t_j}{CT} \right\rceil \qquad \forall\ i = 1, ..., N \tag{13}$$

$$FTA = \left\{(i, j) : j = E_i, ..., L_i\right\} \qquad \forall\ i = 1, ..., N \tag{14}$$

Equations 15 and 16 are the proposed integer adaptation of the reduction strategy. These expressions also consider the number of copies of each task. Note that the bounds are slightly softer for the integer formulation. E_i^*, in the integer formulation, means that at least one copy of task i can be allocated in such station, but not necessarily all the copies. This bound can be strengthened by also defining an upper bound of the number of copies that can be assigned to a station.

$$E_i^* = \left\lceil \frac{t_i + \sum_{j \in P_i^*} t_j \cdot n_j}{CT} \right\rceil \qquad \forall\ i = 1, ..., N \tag{15}$$

$$L_i^* = M - 1 + \left\lceil \frac{t_i + \sum_{j \in F_i^*} t_j \cdot n_j}{CT} \right\rceil \qquad \forall\ i = 1, ..., N \tag{16}$$

The maximal amount of task's copies can be also determined by counting the duration time of predecessors and successors. The theoretical amount of copies of task i that can be allocated in or before (after) station j is defined as $n_{B_{ij}}$ ($n_{A_{ij}}$). These values are used to calculate the maximal number of copies that can be allocated in each station $n_{max_{ij}}$. In Equation 17, $n_{B_{ij}}$ is defined as the amount of copies that can be fit in a station after the processing time of all its predecessors are considered. Similarly, Equation 18 defines $n_{A_{ij}}$ based on the follower tasks.

The upper bound for the number of copies of task i in station j ($n_{max_{ij}}$) is defined in the interval $[E_i^*; L_i^*]$. This bound is determined by the minimal value between the number of available copies of the task (n_i), the amount of tasks an empty station can support $\left\lfloor \frac{CT}{t_i} \right\rfloor$, and the amount of tasks that fit in the station when one also considers the predecessors ($n_{B_{ij}}$) and successors ($n_{A_{ij}}$), as seem in Equation 19.

$$n_{B_{ij}} = \left\lfloor \frac{j \cdot CT - \sum_{P_i^*} n_k \cdot t_k}{t_i} \right\rfloor \tag{17}$$

$$n_{A_{ij}} = \left\lfloor \frac{(M + 1 - j) \cdot CT - \sum_{F_i^*} n_k \cdot t_k}{t_i} \right\rfloor \tag{18}$$

$$n_{max_{ij}} = \min\left(n_i, \left\lfloor \frac{CT}{t_i} \right\rfloor, n_{B_{ij}}, n_{A_{ij}}\right) \tag{19}$$

An example of the use of the earliest and latest stations can be seen in Table 1. The table shows the FTA set for the instances represented in Figures 2-4. Note that the Binary-Vertical's (Fig. 2) tasks have the most assignment possibilities. The ordering of identical tasks performed to create the Binary-Horizontal (Fig. 3) instance reduces the assignment possibilities, once more precedence relations are considered. The FTA set for the integer instance is identical to the Binary-Vertical (Bin-V) set. With the information of $n_{max_{ij}}$, however, the variable reduction presents the same information of the Binary-Horizontal (Bin-H) problem. Tasks 1 and 2, for example, can only be assigned to the first station for the Binary-H instance while Task 3 would fit in either station 1 or 2. The same information is given in the Integer instance (Int) by allowing at most 3 copies of Task 1 in Station 1 (Binary Tasks 1, 2, and 3) and 1 copy in Station 2 (Binary Task 3).

Table 1: Feasible Task Allocations (FTA) for the precedence diagram of Figures 2, 3, and 4 using the equations 12-16 and 25 as an estimative to CT. For the binary instances, an "x" represent a feasible assignment for a task in a station. For the integer instance, feasible task-station allocations are represented by cells containing numbers. Those numbers represent the maximal number of copies that can be assigned to each station ($n_{max_{ij}}$).

		Tasks														
Instance	Stations	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
Bin-V	1	x	x	x	x	x	x	x	x	x						
	2	x	x	x	x	x	x	x	x	x	x	x	x			
	3				x	x	x	x	x	x	x	x	x	x	x	x
	4													x	x	x
		01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
Bin-H	1	x	x	x	x			x	x							
	2			x	x	x	x	x	x	x	x	x	x			
	3					x	x	x	x	x	x	x	x	x	x	
	4														x	x
			1			2			3			4			5	
Int	1		3			1			2							
	2		1			3			3			3				
	3					2			3			3			2	
	4														2	

4. Computational Experiments

This section explores the differences in performance of the binary and integer formulations on a series of tests based on benchmark and real inspired instances. Both integer and binary formulations were implemented on a state-of-the-art universal solver. The tests were solved on a 2.3 GHz, Intel i7-3610 QM, computer with 16.0 GB of RAM. A specialized algorithm was also used in order to have a performance reference. As reported by the comparative study of Pape (2015), and the survey from Scholl & Becker (2006), branch-and-bound procedures that base the search procedure on the problem structure are among the best solution methods for solving SALBP. Out of the available alternatives, SALOME is the best procedure capable of directly solving the type-2 problem. Therefore, a version of SALOME-2 (Klein & Scholl, 1996), obtained

in http://www.assembly-line-balancing.de, was used for the comparison.

As seen in Section 2, a SALBP instance with multiple identical tasks can be simplified by applying Jackson's dominance rule. For the binary-based formulation, both variants are tested: problems with no precedence relations between identical tasks as well as one with a fixed order of assignment were created from each problem. The variations with no extra precedence relations, as exemplified in Fig. 2, are named Binary-Vertical (*Bin-V*) case, while the cases based on Fig. 3 are called Binary-Horizontal (*Bin-H*).

Three sets of instances are proposed to test the formulations performance. Subsection 4.1 presents 9 precedence diagrams from the standard dataset from Scholl (1999) that are adapted to create instances with multiple identical tasks. In Subsection 4.2, the effect of the number of copies is tested with 121 instances of 2 precedence diagrams ranging from 1 copy of each task to enough copies to result in a problem with up to 1000 tasks. Three instances inspired in a real spot welding line are described in are described in Subsection 4.3 and a review on the model's characteristics and results is shown in Subsection 4.4.

4.1. Adapted Standard Dataset

The welding operations observed by the authors in real automobile assembly lines presented tasks ranging from a single operation up to 50 identical welding points. The average number of identical tasks in observed vehicles is between the interval from 7 to 12 copies. For the given study, the generated instances are created with 10 copies of each task. In order to create a dataset with instances that are similar in size to Scholl's benchmark, 9 precedence diagrams are adapted to the 225 instances used in the test. Table 2 summarizes the precedence diagrams used, the number of tasks of each instance and the range of stations used for each case.

Table 2: Selected instances of Scholl's benchmark. Each problem was adapted so that every task presents 10 copies. The problems range from 70 to 350 tasks.

Instances	Tasks	Range of Stations
Bowman	80	4 - 16
Gunther	350	17 - 50
Jackson	110	5 - 22
Jaeschke	90	4 - 18
Heskia	280	14 - 56
Mansoor	110	5 - 22
Mertens	70	3 - 14
Mitchell	210	10 - 42
Roszieg	250	12 - 50

The tests are performed with a time limit of 300 seconds using both the universal solver and SALOME-2. The integer formulation (Subsection 3.2) is represented by *Int-Model*, while *Bin-H Model* and *Bin-V Model* represents Patterson & Albracht (1975) formulation applied to Binary-Horizontal (e.g. Figure 3) and Binary-Vertical (e.g. Figure 2) versions of the instance, respectively. Both binary instances are also solved using SALOME and are represented by *SALOME-H* and *SALOME-V*. For the cases solved with the presented model (Subsections 4.1 - 4.3), a sample run

with 15 seconds of processing was used as an upper bound for the cycle time in the variable set reduction presented on Subsection 3.3.

Table 3 summarizes the results. The horizontal and vertical forms of organizing the task copies presented considerable influence in the algorithms' performance. Even thought SALOME uses dominance rules, the two variations of the problems are not identical. The algorithm might still compare several equally good partly solutions when identical tasks don't present precedence relations between them (for Binary-Vertical alignment). On the other hand, the binary formulation solved by the universal solver showed an opposite trend. The mixed-integer linear programming method of resolution used in the solver seemed to evaluate better the identical solutions and reached superior answers for the *Bin-V* cases (see columns $\#OF$ and $\#OP$). Fixing an operation order produces a more restricted problem and, therefore, resulted in instances in which the binary model was unable even to find a feasible integer solution, indicated in column $\#Fail$.

Table 3: Computational results for the 225 adapted instances of Section 4.1. #OF stands for the number of optimum found, #OP for the optimum proven, #Fail the number of instances without a feasible solution found. The average deviation from optimum is shown in $\varnothing dev.$ column, while $\varnothing Time$ accounts for the average processing time. Time was assumed to be equal to the time limit (300 seconds) when a fail occurred.

Cases	#OF	#OP	#Fail	$\varnothing dev.$	$\varnothing Time(s)$
SALOME-H	219	218	0	0.079%	10.6
SALOME-V	203	200	0	0.287%	41.0
Int-Model	189	186	0	0.196%	58.7
Bin-H Model	76	71	68	54.76%	212.8
Bin-V Model	125	94	0	16.13%	185.4

The integer formulation brought significant improvement in the number of optimal solutions found when compared to the binary models. With the time limit of 5 minutes, the integer model was able to prove optimality of almost twice as much as the best of the binary formulations (column $\#OP$, 186 versus 94), while reducing relative deviation from optimal from 16.13% (Bin-V) or 54.76% (Bin-H) to just 0.196%. Although the integer model, for this testbed, was inferior than SALOME's performance, the results were more similar to SALOME than the binary formulations. It is important to highlight that the Integer Model is solved by a general purpose solver, while SALOME is a specialized algorithm for this kind of problem.

4.2. Effect of the Number of Copies

To study the formulations' performance influenced by the number of copies of each task, two precedence diagrams from the dataset of Scholl (1999) were chosen. Both Mansoor and Lutz1 cases present few tasks, allowing a big varieties of number of copies to be tested, while their solution still proved to be challenging. The maximal number of copies was determined by SALOME's limit of 1000 tasks. Mansoor's precedence diagram contains 11 tasks, resulting in 90 instances ranging from 1 to 90 copies up to 990 tasks, while Lutz1's diagram (32 tasks) resulted in instances from 1 to 31 copies.

The number of workstations used for each instance is calculated by Equation 20. A function of the number of copies (NC) is used to determine an adequate number of stations (NS) to create challenging instances, as indicated in Tables 4 and 5. For the Mansoor's cases the coefficient NS_0 is set to 3, while $NS_0 = 7$ for the Lutz1's problems.

$$NS = \left\lceil NS_0 \cdot NC^{(3/4)} \right\rceil \tag{20}$$

The results are presented in Tables 4 and 5. The instances are grouped according to the number of copies of each task in 6 intervals. For instance, in Table 4 the intervals are 01 to 15 copies, 16 to 30 copies, and so on. The output data presented is the sum for the time limit of 3,600 seconds. For both cases, SALOME-H proved more optimal answers ($\#OP$) than the other methods: 77 out of 90 for Mansoor's diagram and 13 out of 31 for Lutz1's problems. The performance of SALOME-V, however, was greatly inferior without the preordering of identical tasks: only 27 (21+6) out of the 121 (90+31) instances had optimal proved solutions. For the instances solved with the universal solver, only the integer formulation was able to give reasonable results. The integer formulation proved 52 out of the 121 instances, a result between the performance levels of SALOME-H and SALOME-V. The binary formulations using the solver underperformed greatly the results of the other methods; only the very small instances were solved to optimality.

Table 4: Results obtained for the Mansoor's precedence diagram. The instances are separated by the number of copies. The methods' results are separated by slashes in the order: SALOME-H/SALOME-V/**Int-Model**/Bin-H Model/Bin-V Model. Each line represents the sum of 15 instances. The number of instances whose optimality has been proven for up to 3,600 seconds of processing is displayed in column #OP. The #BF (Best incumbent Found) column represents for how many instances which method obtained the best solution. The sum of the difference of the answer obtained and the best result is displayed in the #Abs. Dev. (Absolute Deviation) column. The solver was not able to get an integer solution for all cases of Bin-H Model. The number of instances considered in the sum are shown between brackets. The last column shows the amount of time each method processed the instances.

Instances	#OP	#BF	#Abs Dev.	Total Time
MANSOOR 01-15	15/15/**15**/4/4	15/15/**15**/8/14	0/0/**0**/49(13)/1	0.06/1,148/**22.6**/39,601/41,690
MANSOOR 16-30	15/2/**8**/0/0	15/2/**10**/0/6	0/52/**9**/-(0)/33	29.8/46,800/**25,364**/54,000/54,000
MANSOOR 31-45	15/0/**8**/0/0	15/1/**10**/0/3	0/75/**6**/4,240(6)/467	2,431/54,000/**26,131**/54,000/54,000
MANSOOR 46-60	15/4/**12**/0/0	15/5/**12**/0/0	0/117/**3**/12,977(14)/4,722	2.0/39,686/**11,039**/54,000/54,000
MANSOOR 61-75	7/0/**0**/0/0	11/1/**7**/0/0	7/84/**14**/9,705(11)/4,441	38,119/54,000/**54,000**/54,000/54,000
MANSOOR 76-90	10/0/**6**/0/0	10/2/**11**/0/0	14/80/**5**/15,236(15)/12,232	18,070/54,000/**32,959**/54,000/54,000
MANSOOR 01-90	77/21/**49**/4/4	81/26/**65**/8/23		

Once several instances were not solved within the time limit, the amount of best answers found from each method ($\#BF$) and the difference of the incumbent found to the best ($\#Abs.Dev$) are also represented in the tables. For the Mansoor's instances, SALOME-H still found most of the best answers (81 out of 90), followed by Int-Model (65 out of 90) and SALOME-V (26 out of 90). In Lutz1's cases, however, the Int-Model reached the best answers in 17 of the 31 cases,

Table 5: Results obtained for the Lutz1's precedence diagram. The methods' results are separated by slashes in the order: SALOME-H/SALOME-V/**Int-Model**/ Bin-H Model/Bin-V Model. The table is organized in the same manner as Table 4. The instances are grouped in set of 5 or 6.

Instances	#OP	#BF	#Abs Dev.	Total Time
LUTZ1 01-05	5/5/**3**/2/1	5/5/**3**/3/3	0/0/**38**/26(5)/34	1.0/11.9/**10,968**/14,111/14,400
LUTZ1 06-10	5/1/**0**/0/0	5/1/**0**/0/0	0/378/**156**/370(4)/142	1,682/14,574/**18,000**/18,000/18,000
LUTZ1 11-15	3/0/**0**/0/0	4/0/**0**/0/1	6/552/**192**/-(0)/262	12,227/18,000/**18,000**/18,000/18,000
LUTZ1 16-20	0/0/**0**/0/0	1/0/**4**/0/0	207/489/**1**/-(0)/3,775	18,000/18,000/**18,000**/18,000/18,000
LUTZ1 21-25	0/0/**0**/0/0	1/0/**4**/0/0	155/391/**28**/-(0)/17,258	18,000/18,000/**18,000**/18,000/18,000
LUTZ1 26-31	0/0/**0**/0/0	0/0/**6**/0/0	334/703/**0**/-(0)/279,812	21,600/21,600/**21,600**/21,600/21,600
LUTZ1 01-31	13/6/**3**/2/1	16/6/**17**/3/4		

while SALOME-H obtained 16 of the best answers and SALOME-V only 6. The individual results for each instance can be observed in Figure 5 for Mansoor's diagram and Figure 6 for Lutz1's diagram. Note that for most of the cases, the Int-Model found better answers than SALOME-V but it is outperformed by SALOME-H for the small instances. For the bigger instances, however, the integer formulation was able to found better answers within the time limit.

Once SALOME is a SALBP dedicated algorithm, its performance is expected to be far superior than one of a universal solver. Although it is the case for the binary formulations solved with the solver, the integer formulation produced competitive results in terms of solution quality, especially for the largest instances.

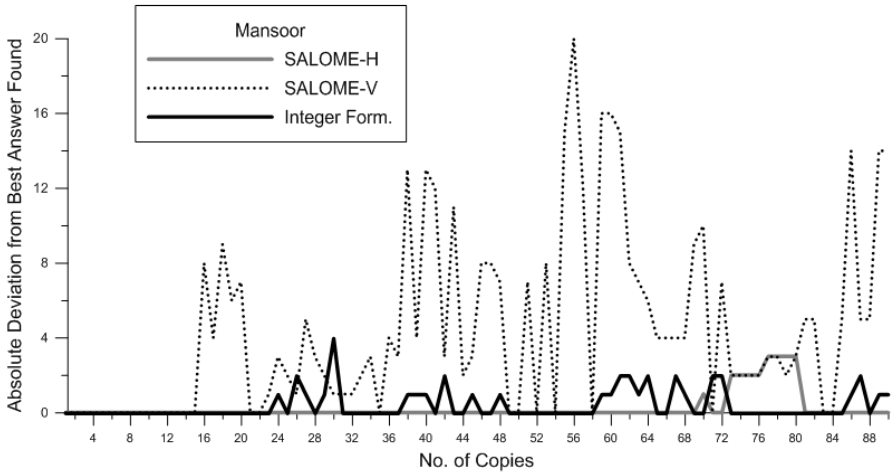


Figure 5: Absolute deviation from the best answer found from each of the methods for Mansoor's instances. Bin-H Model and Bin-V Model's deviations are not shown due to the scale of the graph.

4.3. Real World Instances

In this section three case studies based on real body welding lines are presented. Two product models with 241 and 248 tasks are produced in a line using 26 robotic workers, while the third model uses 28 robots to distribute its 312 welding points.

The Figure 7 shows the precedence diagram of the process. Each node represents one type of task and the diagram is the same for all three models. The products differ only in the number of

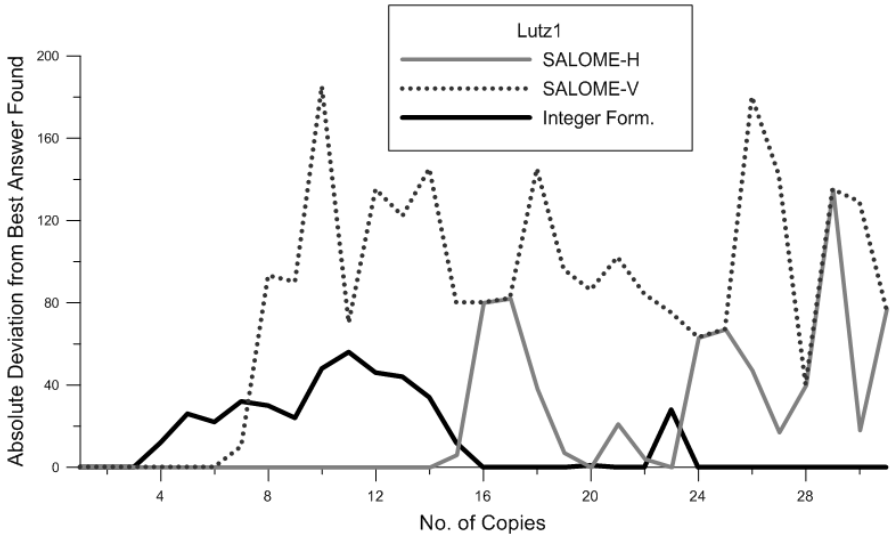


Figure 6: Absolute deviation from the best answer found from each of the methods for Lutz1's instances. Bin-H Model and Bin-V Model's deviations are not shown due to the scale of the graph.

copies and the processing time of each task. The welding spots are performed either to assemble pieces or to reinforce the structure. Structure reinforcements are independent from each other and therefore do not need precedence relations (Tasks 4-9, 12-15, 18, 21-23). Welding spots used to unite parts (Tasks 1-3, 10-11, 16-17, and 19-20), on the other hand, must be performed in a specific order. Furthermore, the extra parts may block the access of the robots to reinforcement spots. The precedence diagram is then shaped as alternating blocks of piece jointing procedures and blocks of reinforcement procedures that must be performed before their accessibility is lost.

Figures 1 and 8 exemplify how the precedence relations are formed. In Figure 1, the filled circles (G_1) represent Task 12, the unfilled circles (G_2) exemplify Task 16, and the asterisks (G_3) stand for Task 18. The group G_2 is also present in Part 2 (Figure 8a) once it represents the union procedures of the two parts. After the parts are assembled, Task 12 is not accessible anymore, as it can be seen in Figure 8b. Therefore, a precedence relation between task 12 and 16 is defined. Task 18 on the other hand, is not blocked by Part 2, so that no precedence relation between Tasks 12 and 18 is defined.

The number of copies and the processing time of each task is summarized in Table 6. Along with the precedence diagram of Figure 7, they define the three proposed case studies. The time required for a welding spot is almost constant. The gun movement, however, causes most of the processing time differences. Tasks with similar accesses were gathered in a single type of task. Their processing time is given as an approximation of the time a robot takes to reach the correct position and perform the operation.

Binary versions of the instances were created both for the horizontal and vertical variants. Table 7 contains the results with a processing limit of 60 seconds comparing the integer formulation to SALOME, while 3,600 seconds was used as a limit for the results in Table 8. For all cases, the integer formulation provided better results. Two of the three case studies were solved with a short

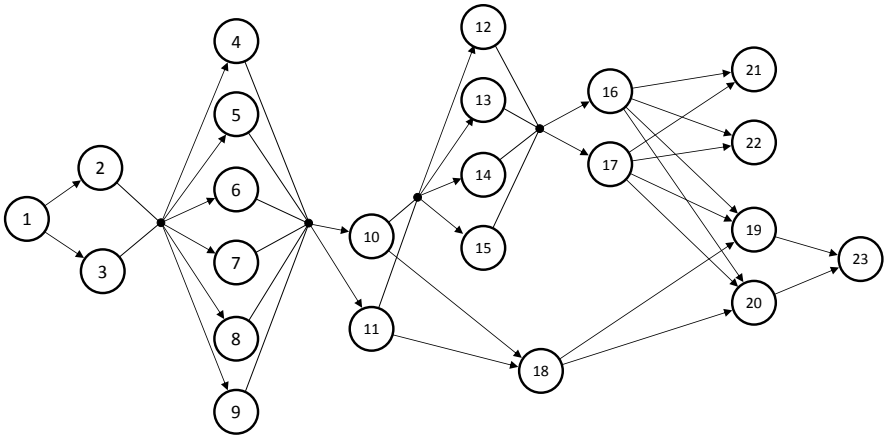
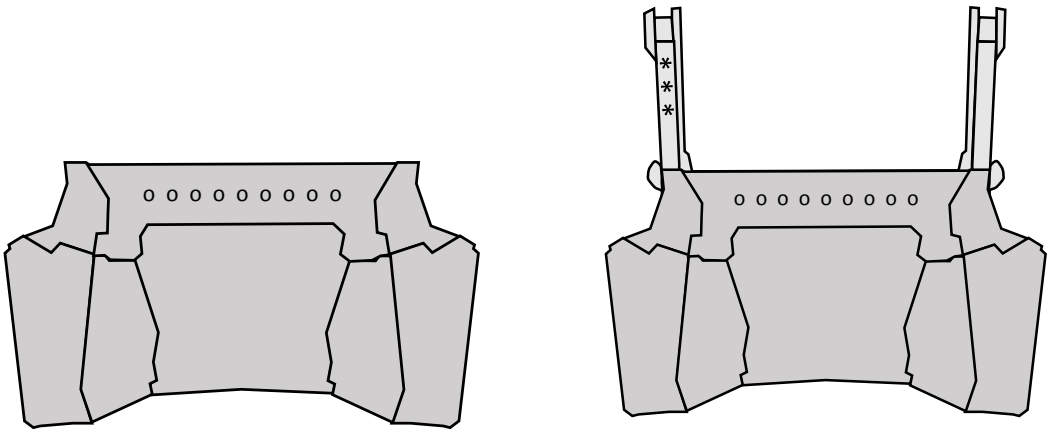


Figure 7: Precedence diagram for the three real-world inspired case studies. To reduce the number of arcs representing precedence relations, intermediary black points are used in the representation. When these points are used, all tasks linked to them precede all tasks they are pointed at. For instance, node 2 precedes nodes 4 to 9.



(a) Part 2 to be assembled to the Part 1 of Figure 1.

(b) Assembled parts.

Figure 8: Representation of the parts assembled by the assembly line in the case study (Figure 1 and 8a). The asterisks and circles represent welding points. Note that after the assemblage, some points of Part 1 became inaccessible to further welding procedures.

Table 6: Number of copies and processing time of each task copy in normalized time units for each case study. Note that not every task of the precedence diagram of Figure 7 is present in every model.

	Model 1		Model 2		Model 3	
Task	Copies	Time	Copies	Time	Copies	Time
1	8	57	8	57	8	57
2	6	38	6	38	6	38
3	6	50	6	50	6	50
4	6	47	4	49	10	42
5	10	29	10	29	20	27
6	14	58	10	57	6	55
7	18	40	18	40	22	43
8	4	47	4	47	8	43
9	4	63	4	63	2	64
10	15	63	15	63	15	63
11	13	39	13	39	13	39
12	7	42	11	38	11	38
13	34	35	46	34	40	37
14	21	70	18	71	37	77
15	11	28	7	26	13	21
16	15	69	15	69	16	71
17	5	44	5	44	8	45
18	0	-	0	-	12	37
19	20	34	6	35	18	32
20	0	-	12	50	4	52
21	12	35	12	35	14	33
22	12	55	12	55	12	51
23	0	-	6	56	11	52
Total	241	11,257	248	11,575	312	14,534

processing time (13.5 and 99.3 seconds) while the best bounds for the Model 1 were obtained by the integer formulation. Again, this formulation simplifies the problem, allowing a universal solver to compete with SALOME for problems with repetitive tasks.

Table 7: Results obtained by the integer formulation and SALOME with a limit of 60 seconds of processing. Values between brackets represent the best incumbent solution found on the right side and the lower bound on the left side.

	Int-Model		SALOME Bin-H		SALOME Bin-V	
Model	Time (s)	Solution	Time (s)	Solution	Time (s)	Solution
1	60	[437;440]	60	[433;447]	60	[433;440]
2	13.5	451	60	[446;453]	60	[446;453]
3	60	[520;521]	60	[520;527]	60	[520;521]

Table 8: Results obtained by the integer formulation and SALOME with a limit of 3,600 seconds of processing. Values between brackets represent the best incumbent solution found on the right side and the lower bound on the left side.

	Int-Model		SALOME Bin-H		SALOME Bin-V	
Model	Time (s)	Solution	Time (s)	Solution	Time (s)	Solution
1	3,600	[437;440]	3,600	[433;447]	3,600	[433;440]
2	13.5	451	2,305.1	451	3,600	[446;453]
3	99.3	521	3,600	[520;527]	3,600	[520;521]

4.4. Formulation's discussion

The integer formulation allows the representation of groups of tasks instead of treating every one individually. This concept allows the use of fewer variables when identical tasks are present. The differences in terms of the number of variables and restrictions of the integer and binary formulations for the real-world inspired instances are showed in Table 9. The integer formulation possesses one integer variable and one binary variable for each type of task, resulting in fewer variables and restrictions when compared to the binary models. Note that the Binary-Vertical formulation has much more restrictions than the Binary-Horizontal formulation. The Binary-Vertical instance has several direct precedence relations, which result in several restrictions. The extra precedence relations added to order the identical tasks in the Binary-Vertical formulation results in a diagram with fewer direct precedence relations, transforming a great number of them in indirect relations. The effect of the variable set reduction (subsection 3.3) can be observed comparing Tables 9 and 10.

As the amount of copies of tasks increases, the number of variables of the binary models tend to increase linearly. On the other hand, the variable set's size of the integer model remains somewhat constant: a new copy, instead of creating new variables, just increases the upper bound integer variables can assume. Although models with integer variables tend to be difficult to solve (Williams, 2009), for the instances presented here, the integer formulation has substantially fewer variables, resulting in a performance comparable to SALOME.

Table 9: Number of variables and restrictions of the integer, binary-horizontal and binary-vertical for the three case studies of Subsection 4.3. The last row contains the upper bound for the cycle time used in the variable set reduction. Half of the variables of the integer formulation is integer. For the binary formulations, only CT is an integer variable.

With Reduction		Model 1	Model 2	Model 3
Integer Form.	Variables	233	243	315
	Restrictions	456	473	622
Binary-H	Variables	1,205	1,578	1,866
	Restrictions	1,606	2,038	2,380
Binary-V	Variables	1,515	1,932	2,465
	Restrictions	42,359	55,068	75,632
Upper Bound for CT		447	453	527

Table 10: Number of variables and restrictions of the integer, binary-horizontal and binary-vertical for the three case studies of Subsection 4.3 without the variable set reduction.

Without Reduction		Model 1	Model 2	Model 3
Integer Form.	Variables	1,041	1,145	1,289
	Restrictions	2,906	3,168	3,607
Binary-H	Variables	6,267	6,449	8,737
	Restrictions	7,313	7,554	10,056
Binary-V	Variables	6,267	6,449	8,737
	Restrictions	183,099	189,866	291,540

5. Conclusion

Identical or repetitive tasks are frequently present in assembly lines that welds parts or fixes components with screws. When considered individually, a pre-ordering can be applied to the identical tasks to simplify the problem. In this paper, an integer formulation is proposed to gather identical tasks and treat them as a group. One integer decision variable per task group is used to determine how many copies of a task are assigned to a workstation instead of using one binary variable for each copy.

The integer and binary formulations were tested using both adapted SALBP-2 instances from a standard dataset and real-world inspired cases of an auto manufacturer company. Using a universal solver, the integer formulation outperformed the binary formulation of Patterson & Albracht (1975) for the vast majority of instances tested. When compared to SALOME (Klein & Scholl, 1996), a specific algorithm designed for SALBP, the integer formulation solved using a generic solver presented results in the same magnitude. No clear dominance between the two methods can be inferred. SALOME solved preordered instances with a shorter processing time and proved the optimality of more instances of the benchmark. The implemented integer formulation was able to find the best answers and bounds for several of the instances with a high number of tasks, including the real-world inspired cases. Although a mixed-integer linear programming solver is not dedicated to SALBP, it showed itself competitive along with the integer formulation when several identical tasks are present. This fact shows that the capability of solving SALBP problems using the proposed formulation is enhanced and therefore larger instances can be solved.

The use of integer variables for the simplified representation of identical tasks can be integrated in other solving methods and is not restricted to SALBP models. The advantages of the integer formulation can be extended to further assembly line balancing problems with different characteristics. Furthermore, an hybrid model using either integer or binary variables depending on the number of copies of a task can result in even more compact and efficient formulations.

References

Battaïa, O., & Dolgui, A. (2013). A taxonomy of line balancing problems and their solution approaches. *International Journal of Production Economics*, *142*, 259–277. doi:10.1016/j.ijpe.2012.10.020.

Bautista, J., & Pereira, J. (2002). Ant algorithms for assembly line balancing. *Lecture Notes in Computer Science*, *2463*, 65–75.

Bautista, J., & Pereira, J. (2009). A dynamic programming based heuristic for the assembly line balancing problem. *European Journal of Operational Research*, *194*, 787–794. doi:10.1016/j.ejor.2008.01.016.

Baybars, I. (1986). 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. doi:10.1016/j.ejor.2004.07.023.

Blum, C. (2008). Beam-ACO for simple assembly line balancing. *INFORMS Journal on Computing*, *20*, 618–627. doi:10.1287/ijoc.1080.0271.

Borba, L., & Ritt, M. (2014). A heuristic and a branch-and-bound algorithm for the Assembly Line Worker Assignment and Balancing Problem. *Computers & Operations Research*, *45*, 87–96. doi:10.1016/j.cor.2013.12.002.

Bowman, E. H. (1960). Assembly-Line Balancing by Linear Programming. *Operations Research*, *8*, 385–389.

Boysen, N., Fliedner, M., & Scholl, A. (2007). A classification of assembly line balancing problems. *European Journal of Operational Research*, *183*, 674–693. doi:10.1016/j.ejor.2006.10.010.

Boysen, N., Fliedner, M., & Scholl, A. (2008). Assembly line balancing: Which model to use when? *International Journal of Production Economics*, *111*, 509–528. doi:10.1016/j.ijpe.2007.02.026.

Falkenauer, E., & Delchambre, A. (1992). A genetic algorithm for bin packing and line balancing. In *Proceedings - IEEE International Conference on Robotics and Automation* (pp. 1186–1192). volume 2.

Fleszar, K., & Hindi, K. S. (2003). An enumerative heuristic and reduction methods for the assembly line balancing problem. *European Journal of Operational Research*, *145*, 606–620. doi:10.1016/S0377-2217(02)00204-7.

Geoffrion, A. M. (1967). Integer Programming by Implicit Enumeration and Balas' Method. *SIAM Review*, *9*, 178–190.

Gökcen, H., & Erel, E. (1998). Binary integer formulation for mixed-model assembly line balancing problem. *Computers and Industrial Engineering*, *34*, 451–461.

Hamzadayi, A., & Yildiz, G. (2013). A simulated annealing algorithm based approach for balancing and sequencing of mixed-model U-lines. *Computers & Industrial Engineering*, *66*, 1070–1084. doi:10.1016/j.cie.2013.08.008.

Jackson, J. (1956). A computing procedure for a line balancing problem. *Management Science*, *2*, 261–271.

Klein, R., & Scholl, A. (1996). Maximizing the Production Rate in Simple Assembly Line Balancing - A Branch and Bound Procedure. *European Journal of Operational Research*, *91*, 367–385.

Kucukkoc, I., & Zhang, D. Z. (2014). Simultaneous balancing and sequencing of mixed-model parallel two-sided assembly lines. *International Journal of Production Research*, *52*, 3665–3687. doi:10.1080/00207543.2013.879618.

Lapierre, S. D., Ruiz, A., & Soriano, P. (2006). Balancing assembly lines with tabu search. (pp. 826–837). volume 168. doi:10.1016/j.ejor.2004.07.031.

Lee, T. O., Kim, Y., & Kim, Y. K. (2001). Two-sided assembly line balancing to maximize work relatedness and slackness. *Computers & Industrial Engineering*, *40*, 273–292. doi:10.1016/S0360-8352(01)00029-8.

Moreira, M. C. O., Cordeau, J.-F., Costa, A. M., & Laporte, G. (2015). Robust assembly line balancing with heterogeneous workers. *Computers and Industrial Engineering*, *88*, 254–263. doi:10.1016/j.cie.2015.07.004.

Nearchou, A. C. (2005). A differential evolution algorithm for simple assembly line balancing. *In16th International Federation of Automatic Control (IFAC) World Congress*, .

Pape, T. (2015). Heuristics and lower bounds for the simple assembly line balancing problem type 1: Overview, computational tests and improvements. *European Journal of Operational Research*, *240*, 32–42.

Pastor, R., & Ferrer, L. (2009). An improved mathematical program to solve the simple assembly line balancing problem. *International Journal of Production Research*, *47*, 2943–2959. doi:10.1080/00207540701713832.

Patterson, J. H., & Albracht, J. J. (1975). Assembly-Line Balancing: Zero-One Programming with Fibonacci Search. *Operations Research*, *23*, 166–172. doi:10.1287/opre.23.1.166.

Ramezanian, R., & Ezzatpanah, A. (2015). Modeling and solving multi-objective mixed-model assembly line balancing and worker assignment problem. *Computers and Industrial Engineering*, *87*, 74–80. doi:10.1016/j.cie.2015.04.017.

Ritt, M., & Costa, A. M. (2015). Improved integer programming models for simple assembly line balancing and related problems. *International Transactions in Operational Research*, *online*. doi:10.1111/itor.12206.

Sabuncuoglu, I., Erel, E., & Tanyer, M. (2000). Assembly line balancing using genetic algorithms. *Journal of Intelligent Manufacturing*, *11*, 295–310. doi:10.1023/A:1008923410076.

Salveson, M. (1955). The assembly line balancing problem. *Journal of Industrial Engineering*, *6*, 18–25.

Scholl, A. (1999). *Balancing and Sequencing of Assembly Lines*. (2nd ed.). Heidelberg: Physica.

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. doi:10.1016/j.ejor.2004.07.022.

Scholl, A., & Klein, R. (1997). SALOME: A bidirectional branch-and-bound procedure for assembly line balancing. *INFORMS Journal on Computing*, *9*, 319–334.

Scholl, A., & Klein, R. (1999). Balancing assembly lines effectively – A computational comparison. *European Journal of Operational Research*, *114*, 50–58. doi:10.1016/S0377-2217(98)00173-8.

Scholl, A., & Voss, S. (1997). Simple assembly line balancing - heuristic approaches. *Journal of Heuristics*, *2*, 217–244.

Sewell, E. C., & Jacobson, S. H. (2012). A Branch, Bound, and Remember Algorithm for the Simple Assembly Line Balancing Problem. *INFORMS Journal on Computing*, *24*, 433–442. doi:10.1287/ijoc.1110.0462.

Sikora, C. G. S., Lopes, T. C., & Magatão, L. (2016). Traveling worker assembly line (re)balancing problem: model, reduction techniques, and real case studies. *European Journal of Operational Research*, . doi:10.1016/j.ejor.2016.11.027.

Thangavelu, S. R., & Shetty, C. M. (1971). Assembly Line Balancing by Zero-One Integer Programming. *AIIE Transactions*, *3*, 61–68. doi:10.1080/05695557108974787.

Vilà, M., & Pereira, J. (2014). A branch-and-bound algorithm for assembly line worker assignment and balancing problems. *Computers & Operations Research*, *44*, 105–114. doi:10.1016/j.cor.2013.10.016.

White, W. W. (1961). Comments on a Paper by Bowman. *Operations Research*, *9*, 274–276.

Williams, H. P. (2009). *Logic and Integer Programming*. Heidelberg London New York: Springer Dordrecht. doi:10.1007/978-0-387-92280-5.

Yolmeh, A., & Kianfar, F. (2012). An efficient hybrid genetic algorithm to solve assembly line balancing problem with sequence-dependent setup times. *Computers and Industrial Engineering*, *62*, 936–945. doi:10.1016/j.cie.2011.12.017.