OR Spectrum (2008) 30:579–609
DOI 10.1007/s00291-006-0070-3

REGULAR ARTICLE

The sequence-dependent assembly line balancing problem

Armin Scholl · Nils Boysen · Malte Fliedner

Published online: 17 November 2006

Abstract Assembly line balancing problems (ALBP) arise whenever an assembly line is configured, redesigned or adjusted. An ALBP consists of distributing the total workload for manufacturing any unit of the products to be assembled among the work stations along the line. The sequence-dependent assembly line balancing problem (SDALBP) is an extension of the standard simple assembly line balancing problem (SALBP) which has significant relevance in real-world assembly line settings. SDALBP extends the basic problem by considering sequence-dependent task times. In this paper, we define this new problem, formulate several versions of a mixed-integer program, adapt solution approaches for SALBP to SDALBP, generate test data and perform some preliminary computational experiments. As a main result, we find that applying SALBP-based search procedures is very effective, whereas modelling and solving the problem with MIP standard software is not recommendable.

Keywords Assembly line balancing · Mass-production · Combinatorial optimization · Sequencing

A. Scholl (✉)
Fakultät für Wirtschaftswissenschaften, Lehrstuhl für Betriebswirtschaftliche Entscheidungsanalyse, Friedrich-Schiller-Universität Jena, Carl-Zeiß-Straße 3, 07743 Jena, Germany
e-mail: a.scholl@wiwi.uni-jena.de

N. Boysen · M. Fliedner
Fakultät Wirtschafts- und Sozialwissenschaften, Institut für Industrielles Management, Universität Hamburg, Von-Melle-Park 5, 20146 Hamburg, Germany
e-mail: boysen@econ.uni-hamburg.de

M. Fliedner
e-mail: fliedner@econ.uni-hamburg.de

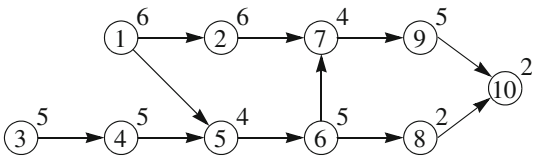

Fig. 1 Precedence graph

1 Introduction

Assembly lines are flow-oriented production systems which are typical in the industrial production of high quantity standardized commodities and even gain importance in low volume production of customized products. Among the decision problems which arise in managing such systems, assembly line balancing problems are important tasks in medium-term production planning (cf., e.g., Baybars 1986; Becker and Scholl 2006; Boysen et al. 2006).

An assembly line consists of (*work*) *stations* $k = 1, \ldots, m$ arranged along a conveyor belt or a similar mechanical material-handling equipment. The workpieces (jobs) are consecutively launched down the line and are moved from station to station. At each station, certain operations are repeatedly performed regarding the *cycle time* (maximum or average time available for each workcycle). The decision problem of optimally partitioning (balancing) the assembly work among the stations with respect to some objective is known as the *assembly line balancing problem* (ALBP).

Manufacturing a product on an assembly line requires partitioning the total amount of work into a set of elementary operations named tasks $V = \{1, \ldots, n\}$ which constitute the nodes of a precedence graph. Performing a task i takes a *task time* (node weight) t_i and requires certain equipment of machines and/or skills of workers. Due to technological and organizational conditions, *precedence relations* between the tasks have to be observed. A precedence relation (i, j) means that task i must be finished before task j can be started and is represented as a directed edge (arc) in the *precedence graph*. Within the arc set E of the graph any arc is removed, which is redundant because it connects nodes that are also connected by a path with more than one arc. Furthermore, we assume that the graph $G = (V, E, t)$ is acyclical and numbered topologically. An example of a precedence graph is given in Fig. 1.

The following sets are useful to describe the precedence relations:

$P_i = \{h | (h, i) \in E\}$ set of direct predecessors of task $i \in V$

$F_i = \{j | (i, j) \in E\}$ set of direct successors (followers) of task $i \in V$

Assuming E^* to be the transitive closure of E, which contains an arc for each path in the precedence graph, we further define:

$P_i^* = \{h | (h, i) \in E^*\}$ set of all predecessors of task $i \in V$

$F_i^* = \{j | (i, j) \in E^*\}$ set of all successors of task $i \in V$

Any type of ALBP consists in finding a feasible *line balance,* i.e., an assignment of each task to a station such that the precedence constraints and possible further restrictions are fulfilled. The set S_k of tasks assigned to a station k $(= 1, \ldots, m)$ constitutes its *station load,* the cumulated task time $t(S_k) = \sum_{j \in S_k} t_j$ is

Table 1 Versions of simple assembly line balancing problem (SALBP)

	cycle time c	
	Given	Minimize
No. (m) of stations		
Given	SALBP-F [||]	SALBP-2 [||c]
Minimize	SALBP-1 [||m]	SALBP-E [||E]

called *station time*. When a fixed common cycle time c is given, a line balance is feasible only if the station time of neither station exceeds c. In case of $t(S_k) < c$, the station k has an *idle time* of $c - t(S_k)$ time units in each cycle, i.e., it is repeatedly unproductive for this time span.

The most popular ALBP is called *simple assembly line balancing problem* (SALBP). It has the following characteristics (cf. Baybars 1986; Scholl 1999, chap. 2.2; Boysen et al. 2006):

- mass-production of one homogeneous product; given production process
- paced line with fixed cycle time c
- deterministic (and integral) operation times t_j
- no assignment restrictions besides the precedence constraints
- serial line layout with m stations
- all stations are equally equipped with respect to machines and workers
- maximize the *line efficiency* $\text{Eff} = t_{\text{sum}}/(m \times c)$ with total task time $t_{\text{sum}} = \sum_{j=1}^{n} t_j$

Several problem versions arise from varying the objective as shown in Table 1. The tuple-notations specify the characterizations of the problem versions within the recent classification scheme of Boysen et al. (2006).

Since SALBP-F is an NP-complete feasibility problem, the optimization versions of SALBP, which may be solved by iteratively examining several instances of SALBP-F, are NP-hard (cf. Wee and Magazine 1982; Scholl 1999, chap. 2.2.1.5).

Recent surveys covering SALBP models and procedures are given by Erel and Sarin (1998), Scholl (1999, chap. 2, 4, 5), Rekiek et al. (2002) as well as Scholl and Becker (2006).

2 The sequence-dependent problem extension

In practice, there is usually a greater flexibility in task times and/or precedence relations than postulated by the traditional precedence graph and assumed by SALBP (and all established extensions of SALBP). Usually, not every precedence relation specified in order to construct a precedence graph is due to a strict technological requirement. Instead, such a relation (i, j) might be specified only because it is assumed to be the more efficient order of performing these tasks.

This becomes relevant whenever the tasks i and j interact in such a manner that their task times are influenced. For example, consider the assembly of a car, where several components have to be installed at the same mounting place or at neighbouring ones (e.g., seat and seat belt, head lights and buffer-bar, motor and cooling system, radio and ventilation device). Installing a seat before the appendant seat belt may prolong the latter task, because the seat is an obstacle which requires additional movements and/or prevents from using the most efficient installation procedure. The other way round, mounting the seat may be restricted by the already installed seat belt resulting in an enlarged task time.

On the one hand, examining apparent precedence relations with respect to this aspect will usually reveal that a considerable number of these constraints are not as hard as considered to be, because they represent a favourable order of performing tasks but not the only possible one. On the other hand, considering the correctness of measured task times will often make evident that they are only valid if the unhindered standard procedure can be used to perform the respective task, i.e., they depend on implicit precedence relations. As this discussion shows, interacting tasks can be modelled neither by specifying strict precedence relations nor by omitting these constraints while the task times are fixed.

In order to model this very typical situation adequately, we introduce the concept of *sequence-dependent task time increments*. Whenever a task j is performed after another task i has been finished, its standard time t_j is incremented by a value sd_{ij}. This sequence-dependent increment measures the prolongation of task j forced by the interference with the status of already having processed task i.

With respect to the direction of interaction, we distinguish two cases:

- *Unidirectional interaction.* If $sd_{ji} > 0$ and $sd_{ij} = 0$, then only task j influences the execution of task i, while task i has no influence on performing task j.
- *Bidirectional interaction.* $sd_{ji} > 0$ and $sd_{ij} > 0$ hold, either task restricts the other.

Example In case of the precedence graph given in Fig. 1, we assume that the task times t_5 and t_6 have been measured without regarding that performing task 2 before would prolong the execution time of task 5 by $sd_{25} = 2$ and that of task 6 by $sd_{26} = 1$ time units due to an interference (unidirectional). Though the precedence relation (6,7) represents the preferable order of performing the tasks 6 and 7, the reverse order is assumed to be possible and connected with an increase $sd_{76} = 1$ of t_6 (unidirectional). Finally, we assume that the tasks 8 and 9 interact bidirectionally such that $sd_{89} = 3$ and $sd_{98} = 1$ have to be added to the standard task times, respectively.

Figure 2 includes this additional aspects by introducing disjunctive arcs (dashed) for all pairs (i, j) of interacting tasks using sd_{ij} and sd_{ji} as arc weights.

Obviously, the tasks i and j only can interact in the described manner if they are not related by precedence, i.e., there is no path in the precedence graph connecting i and j. The set of interacting pairs is called SD, i.e.,

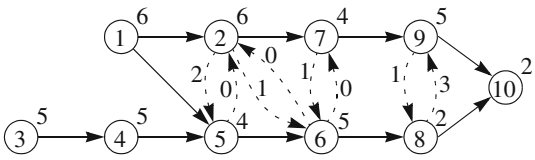

Fig. 2 Precedence graph plus disjunctive arcs

Table 2 Versions of sequence dependent assembly line balancing problem (SDALBP)

	Cycle time c	
	Given	Minimize
No. (m) of stations		
Given	SDALBP-F [Δt_{ind} | |]	SDALBP-2 [Δt_{ind} | | c]
Minimize	SDALBP-1 [Δt_{ind} | | m]	SDALBP-E [Δt_{ind} | | E]

$\text{SD} = \{(i,j) \in (V \times V) - E^* | i < j \wedge \text{sd}_{ij} + \text{sd}_{ji} > 0\}$. The tasks i interacting with a given task j are collected in the set $\text{SD}_j = \{i | (i,j) \in \text{SD} \vee (j,i) \in \text{SD}\}$. If several time increments are to be considered for a task their sum is added to the standard task time (additive time increments).

Whenever there are bidirectional task interactions, the total task time necessarily exceeds the sum of standard times t_{sum}. As a simple lower bound on the modified total task time we get

$$t'_{\text{sum}} = t_{\text{sum}} + \sum_{(i,j) \in \text{SD}} \min\{\text{sd}_{ij}, \text{sd}_{ji}\} \tag{1}$$

Example For the instance of Fig. 2, we get $t'_{\text{sum}} = 44 + \min\{1,3\} = 45$.

Introducing the sequence-dependent task time increments ($\text{SD} \neq \emptyset$) into SALBP leads to a modified problem which we call the *sequence-dependent assembly line balancing problem* (SDALBP). As in case of SALBP, different problem versions can be defined, the names and classification tuples (cf. Boysen et al. 2006) of which are given in Table 2.

In contrast to SALBP-E, there is no obvious way to define the line efficiency, because the total task time depends on the line balance considered. In order to get a solution-independent and realistic measure, we define the line efficiency as $\text{Eff}' = t'_{\text{sum}}/(m \times c)$, because t'_{sum} represents the most efficient way of performing the tasks with respect to their total time requirement.

In order to avoid misunderstandings, some remarks seem to be helpful:

- A solution-dependent definition of the line efficiency using the realized total task time is meaningless, because task orderings which lead to line balances with the same combination of m and c as other orderings but cause unnecessary time increments would increase the line efficiency in an absurd manner.
- The ordering of all interacting task pairs as implied by t'_{sum} need not lead to the most efficient line balance as is shown in the example below.

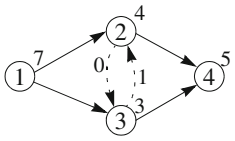

Fig. 3 Difference between simple assembly line balancing problem (SALBP) and sequence-dependent assembly line balancing problem (SDALBP)

- Furthermore, this ordering need not be feasible due to cycles in the induced precedence graph.

Example Taking $c = 11$ and $m = 5$ as parameters of SDALBP-F, a feasible line balance is given by $(S_1 = \{3,4\}, S_2 = \{1,5\}, S_3 = \{6,2\}, S_4 = \{7,9\}, S_5 = \{8,10\})$.[1] In contrast to the SALBP-F solution (cf. Sect. 1), the load $S_4 = \{7,8,9\}$ is not feasible, because the station time is increased to $t(S_4) = t_7 + t_9 + t_8 + min\{sd_{89}, sd_{98}\} = 12$, even if we take the preferred order (7, 9, 8) which constitutes the minimum. Given $c = 11$, the optimal SDALBP-1 solution requires $m^* = 5$ stations (one of the optimal line balances is given above), while SALBP-1 gets by with four stations (because it does not attend that it is not possible to realize both t_8 and t_9 simultaneously). The optimal cycle time for the SDALBP-2 instance given by $m = 5$ is $c^* = 10$. One of the optimal balances is ({1}, {3,4}, {5,2}, {6,7}, {9,8,10}). Given a SDALBP-E instance with the number of stations restricted to 4 or 5, the maximal line efficiency $Eff'^* = 45/(5 \times 10) = 0.9$ is achieved by the SDALBP-2 solution given above, because the minimal cycle time for $m = 4$ stations is $c^* = 14$ with the lower efficiency $45/(4 \times 14) = 0.804$.

In order to make clear the fundamental difference between SALBP and SDALBP unambiguously, we consider only a single pair (i, j) of a unidirectional task interaction ($sd_{ji} > 0$ and $sd_{ij} = 0$). Then, the most intuitive idea consists in introducing the precedence relation (i, j), because this is the preferred task-ordering without incrementing task times at all. However, this may prevent from finding the optimal solution as can be seen by considering the instance defined by Fig. 3 and $c = 10$. If we introduce the arc (2,3), the optimal SDALBP-1 solution ($\{1,3\}, \{2,4\}$) with two stations is not found anymore, because the resulting SALBP-1 instance requires 3 stations ($\{1\}, \{2,3\}, \{4\}$). In this case, it is profitable to accept the time increase $sd_{32} = 1$ in order to allow for combining the tasks 1 and 3.

Since SDALBP is a generalization of SALBP (setting all sd_{ij} to 0 reduces SDALBP to SALBP), its feasibility version SDALBP-*F* is NP-complete and its optimization versions are NP-hard, too.

In the above description and throughout the whole paper, we assume an additive linear influence of increments on the task time, i.e., the actual time of a task j is computed by summing up its original time t_j and all increments sd_{ij} of tasks i which precede j. Sometimes, this might be unrealistic, because the joint influence of several tasks on the time of j might differ from the sum of

[1] Notice that performing task 6 before task 2 in station 2 leads to the feasible station time 11, whereas the reverse order would constitute the infeasible station time 12.

individual influences. In this case, the models and solution procedures have to consider task time increments sd_{Sj} of subsets $S \subseteq V - \{j\} - F_j^* - P_j^*$ instead of sd_{ij} with tasks $i \in V - \{j\} - F_j^* - P_j^*$ only. Because this would complicate the further analysis considerably, we restrict our presentation to the additive linear case. However, notice that incorporating general time increments is possible, albeit rather complex, in the models presented in Sect. 3 and easy in case of the search procedures presented in Sect. 4.

3 Mathematical models for SDALBP

For SALBP, there is a number of integer linear programs available (cf. Scholl 1999, chap. 2.2; Ugurdag et al. 1997; Pinnoi and Wilhelm 1997; Bockmayr and Pisaruk 2001; Peeters and Degraeve 2006). In the following, we develop mixed-integer programs (MIP) which are restricted to the problem version SDALBP-1, because adapting this model to the other problem versions is straightforward (cf. Scholl 1999, chap. 2.2).

3.1 Definitions

The models are developed by extending the standard formulation for SALBP-1 (cf. Bowman 1960; White 1961; Thangavelu and Shetty 1971; Patterson and Albracht 1975; Scholl 1999, p. 29) which is based on *binary assignment variables* x_{jk} defined as follows ($\bar{m}$ is a valid upper bound on the number of stations; a very simple bound is $\bar{m} = n$, for further bounds see Scholl 1999, chap. 2.2.2.2):

$$x_{jk} = \begin{cases} 1 & \text{if task j is assigned to station } k \\ 0 & \text{otherwise} \end{cases} \qquad \text{for } j \in V \text{ and } k = 1, ..., \bar{m}$$

The number of variables can be reduced by computing earliest and latest stations based on the *relative task times* $\tau_j = t_j/c$ of the tasks $j = 1, \ldots, n$, i.e., the portion of the cycle time required by the tasks. In case of SALBP-1, the earliest station E_j and the latest station L_j, respectively, to which a task j can be assigned feasibly is computed as follows (cf. Saltzman and Baybars 1987):

$$E_j = \left\lceil \tau_j + \sum_{h \in P_j^*} \tau_h \right\rceil, \quad L_j = \bar{m} + 1 - \left\lceil \tau_j + \sum_{h \in F_j^*} \tau_h \right\rceil \qquad \text{for } j \in V \tag{2}$$

Thus, task j can only be assigned to one of the stations in the *station interval* $SI_j = [E_j, L_j]$. The other way round, only a subset $B_k = \{j \in V \mid k \in SI_j\}$ of the tasks are *potentially assignable* to the stations $k = 1, \ldots, \bar{m}$.

Since the station intervals are valid for SDALBP-1, too, we can use them to reduce the number of variables such that x_{jk} has only to be defined for $j \in V$ and

$k \in \mathrm{SI}_j$. However, the definitions of E_j and L_j do not consider the sequence-dependent task time increments and might be strengthened accordingly.

The order in which the interacting tasks are executed is represented by *binary ordering variables* y_{ij}:

$$y_{ij} = \begin{cases} 1 & \text{if task } i \text{ is executed prior to task } j \\ 0 & \text{if task } i \text{ is executed after task } j \end{cases} \quad \text{for } (i,j) \in \mathrm{SD}$$

In order to improve the readability and the comprehensiveness of the model, we additionally introduce auxiliary variables z_j replacing a more complex term that determines the index of the station to which task j is assigned:

$$z_j := \sum_{k \in \mathrm{SI}_j} k \times x_{jk} \quad \text{for } j \in V$$

The assumption that n is a single sink node of the precedence graph allows for a simple determination of the number of stations actually required ($m = z_n$). If there are several sink nodes, a fictitious sink node with task time 0 is defined as successor of all original sink nodes and given the (increased) label n.

3.2 Nonlinear model

A *nonlinear mixed-binary* program (*model NL*) is given by the objective function (3), which minimizes the number of stations m, and the constraint set (4)–(16).

$$\text{Minimize } m(\mathbf{x},\mathbf{y},\mathbf{z}) = z_n \tag{3}$$

s.t.

$$\sum_{k \in \mathrm{SI}_j} x_{jk} = 1 \qquad \text{for all } j \in V \tag{4}$$

$$\sum_{j \in B_k} \left(t_j + \sum_{\substack{i \in \mathrm{SD}_j \\ i<j}} \mathrm{sd}_{ij} \times y_{ij} + \sum_{\substack{i \in \mathrm{SD}_j \\ i>j}} \mathrm{sd}_{ij} \times (1 - y_{ji}) \right) \times x_{jk} \le c \quad \text{for } k = 1, \ldots, \bar{m} \tag{5}$$

$$z_i - z_j \le 0 \qquad \text{for all } (i,j) \in E \text{ with } L_i \ge E_j \tag{6}$$

$$z_i - z_j \le \mathrm{M} \times (1 - y_{ij}) \qquad \text{for all } (i,j) \in \mathrm{SD} \tag{7}$$

$$z_j - z_i \le \mathrm{M} \times y_{ij} \qquad \text{for all } (i,j) \in \mathrm{SD} \tag{8}$$

$$y_{ik} \le y_{ij} + y_{jk} \le y_{ik} + 1 \qquad \text{for all } \{i,j,k\} \subset V \textit{ with } (i,j),(j,k),(i,k) \in \mathrm{SD} \tag{9}$$

$$y_{jk} \le y_{ik} \qquad \text{for all } \{i,j,k\} \subset V \text{ with } (i,j) \in E^*, (j,k), (i,k) \in \mathrm{SD} \tag{10}$$

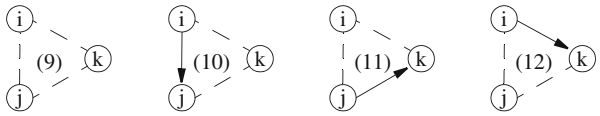

Fig. 4 Triplets of interacting tasks

$$y_{ij} \le y_{ik} \qquad \text{for all } \{i,j,k\} \subset V \text{with} (i,j), (i,k) \in \text{SD}, (j,k) \in E^* \tag{11}$$

$$y_{ij} + y_{jk} \ge 1 \qquad \text{for all } \{i,j,k\} \subset V \text{ with} (i,j), (j,k) \in \text{SD}, (i,k) \in E^* \tag{12}$$

$$z_j = \sum_{k \in \text{SI}_j} k \times x_{jk} \qquad \text{for all } j \in V \tag{13}$$

$$y_{ij} \in \{0,1\} \qquad \text{for all } (i,j) \in \text{SD} \tag{14}$$

$$x_{jk} \in \{0,1\} \qquad \text{for } j \in V \text{ and } k \in \text{SI}_j \tag{15}$$

$$z_j \ge 0 \qquad \text{for } j \in V \tag{16}$$

The constraints (4) assure that each task i is assigned to exactly one station. The nonlinear constraints (5) represent the cycle time restrictions. The left-hand side computes the station time $t(S_k)$ by summing up the actual task times t_j of all tasks j assigned to station k (indicated by $x_{jk} = 1$). The actual task time of a task j is given by the standard task time t_j which is increased by the time increment sd_{ij} whenever a task $i \in \text{SD}_j$ is performed prior to j (indicated by $y_{ij} = 1$ in case of $i < j$ and $y_{ji} = 0$ in case of $i > j$). The resulting station time must be no larger than the cycle time c.

The precedence relations are expressed by the constraint set (6) which ensures that no task is assigned to an earlier (lower-labelled) station than any of its predecessors. This must only be taken into account if the station intervals overlap.

The constraints (7) and (8) determine the order of performing the interacting task pairs. Given a task pair $(i,j) \in \text{SD}$, the corresponding constraint pair (7) and (8) is disjunctive due to a sufficiently large number M (e.g., $\text{M} = \bar{m}$) at the right-hand sides which makes one of both constraints redundant (cf. Williams 1999, chap. 9). The constraint (7) is nonredundant only if the task i is performed prior to j ($y_{ij} = 1 \Rightarrow z_i \le z_j$), whereas the constraint (8) is relevant only when j is executed first ($y_{ij} = 0 \Rightarrow z_i \ge z_j$).

Constraints (9)–(12) ensure that the ordering defined for the subset of interacting tasks is unique. This is achieved by *transitivity* (*anti-cycle*) *conditions* for all subsets with three tasks that are subject to two interaction pairs and one (direct or indirect) precedence constraint or three interaction pairs. Figure 4 shows the four types of task triplets to be considered. The interactions are symbolized by dashed lines, the direct or indirect precedence relations by solid arcs.

The constraint type (9) applies for three tasks $i < j < k$ each pair of which is interacting. In order to get a unique ordering, it is necessary to avoid $y_{ij} = y_{jk} = y_{ki} = 1$ and $y_{ji} = y_{ik} = y_{kj} = 1$, because this would imply a cycle (intransitivity). The first case is avoided by $y_{ij} + y_{jk} + y_{ki} \le 2$ which must be rearranged

to $y_{ij} + y_{jk} \leq y_{ik} + 1$ using the trivial relation $y_{ik} = 1 - y_{ki}$, because y_{ki} is not defined in the model. The second case is avoided by imposing $y_{ji} + y_{ik} + y_{kj} \leq 2$ which is transformed into $(1 - y_{ij}) + y_{ik} + (1 - y_{jk}) \leq 2 \Rightarrow y_{ik} \leq y_{ij} + y_{jk}$.

The constraint types (10)–(12) are simplified versions of (9), because $y_{ij} = 1$, $y_{jk} = 1$, and $y_{ik} = 1$ are induced by the precedence relations, respectively.

Further conditions with four or more tasks are redundant, because each (sub)graph containing more than three tasks covers several subgraphs with three tasks each of which has a feasible ordering and is feasibly connected to another such subgraph by joint task pairs.

The constraints (13)–(16) define and constrain the variables as explained above.

3.3 Linearized model

In order to transform the nonlinear constraints (5) into linear ones, we introduce additional binary variables w_{ijk} defined for all $j \in V, i \in \mathrm{SD}_j$, and $k \in \mathrm{SI}_j$ as follows:

$$w_{ijk} = \begin{cases} 1 & \text{if task } j \text{ is assigned to station } k \text{ and exectued after task } i \\ 0 & \text{otherwise} \end{cases}$$

The constraints (5) and (14) are replaced by the following set of constraints to define a *linear mixed-binary program* for SDALBP-1 [*model L1* = (3)–(24) except for (5) and (14)]:

$$\sum_{j \in B_k} \left(t_j \times x_{jk} + \sum_{i \in \mathrm{SD}_j} \mathrm{sd}_{ij} \times w_{ijk} \right) \leq c \quad \text{for } k = 1, \ldots, \bar{m} \tag{17}$$

$$x_{jk} + y_{ij} \leq w_{ijk} + 1 \quad \text{for } (i,j) \in \mathrm{SD} \text{ and } k \in \mathrm{SI}_j \tag{18}$$

$$x_{jk} \leq w_{ijk} + y_{ji} \quad \text{for } (j,i) \in \mathrm{SD} \text{ and } k \in \mathrm{SI}_j \tag{19}$$

$$w_{ijk} \leq x_{jk} \quad \text{for } j \in V, i \in \mathrm{SD}_j, \text{and } k \in \mathrm{SI}_j \tag{20}$$

$$\sum_{k \in \mathrm{SI}_j} w_{ijk} + \sum_{k \in \mathrm{SI}_i} w_{jik} = 1 \quad \text{for } (i,j) \in \mathrm{SD} \tag{21}$$

$$y_{ij} = \sum_{k \in \mathrm{SI}_j} w_{ijk} \quad \text{for } (i,j) \in \mathrm{SD} \tag{22}$$

$$w_{ijk} \in \{0,1\} \quad \text{for } j \in V, i \in \mathrm{SD}_j, \text{ and } k \in \mathrm{SI}_j \tag{23}$$

$$y_{ij} \geq 0 \quad \text{for } (i,j) \in \mathrm{SD} \tag{24}$$

The constraints (17) increase the standard time of the stations k by the increment sd_{ij}, whenever $w_{ijk} = 1$ signals that task j is assigned to k and the task $i \in \mathrm{SD}_j$ is performed prior to j (in the same or an earlier station). The constraints (18)–(20) enforce $w_{ijk} = 1$ in case of $x_{jk} = 1$ and $y_{ij} = 1$. The conditions

(21) ensure that there is a unique ordering for each interaction pair (i, j). By the Eqs. (22), the ordering variables y_{ij} are set to 1 iff there is a station $k \in \mathrm{SI}_j$ with $w_{ijk} = 1$. Due to (21) this automatically leads to $y_{ji} = 1 - y_{ij}$ such that the ordering variables have still to be defined only for $i < j$. Because the w_{ijk} are binary variables (23), we can relax the integrality condition for y_{ij} in (24). Moreover, the variables y_{ij} could be removed from the model completely by replacing them in all constraints using (22).

Because this is true for the station number variables z_j , too, the only variables required are the assignment variables x_{jk} and the combined ordering and assignment variables w_{ijk}. That is, we have up to $n^2 + 2 \times |\mathrm{SD}| \times n$ binary variables. However, this upper limit is reached only if the weak bound $\bar{m} = n$ is used.

The number of constraints is bounded from above by $2n + |E| + |\mathrm{SD}| \times (3+4n)$ for all constraints except the transitivity conditions (9)–(12), the maximum number of which is given by the number of triplets $\frac{1}{6} \times n^3 - \frac{1}{2} \times n^2 + \frac{1}{3} \times n$ of n tasks in case that all task pairs are interacting.[2]

3.4 Alternative modelling of transitivity constraints

The potentially large number of transitivity constraints can be replaced by only $2n$ constraints by slightly redefining the variables z_j, which then become necessary.

Using a sufficiently small constant $\varepsilon \in (0, 1)$, the conditions (6)–(13) are substituted by the following constraints such that we get *model L2* defined by (3), (4), and (15)–(29):

$$z_i - z_j \leq -\varepsilon \qquad \text{for all } (i,j) \in E \text{ with } L_i \geq E_j \qquad (25)$$

$$z_i - z_j \leq M \times (1 - y_{ij}) - \varepsilon \times y_{ij} \qquad \text{for all } (i,j) \in \mathrm{SD} \qquad (26)$$

$$z_j - z_i \leq M \times y_{ij} - \varepsilon \times (1 - y_{ij}) \qquad \text{for all } (i,j) \in \mathrm{SD} \qquad (27)$$

$$z_j \geq \sum_{k \in \mathrm{SI}_j} k \times x_{jk} \qquad \text{for all } j \in V \qquad (28)$$

$$z_j \leq \sum_{k \in \mathrm{SI}_j} (k \times x_{jk}) + 1 - \varepsilon \qquad \text{for all } j \in V \qquad (29)$$

The effect is as follows: the integral part of z_j still identifies the number of the station, the task j is assigned to. The real-valued part enforces a unique ordering of the subset of tasks within the station load which are related to each other by precedence or interaction pairs. As a sufficiently small constant, we may use $\varepsilon = 1/n$.

[2] See Lübke (2006). Due to the relation $|E| + |\mathrm{SD}| \leq \frac{n \times (n-1)}{2}$, the number of constraints is of order $O(n^3)$.

4 Solution approaches to SDALBP

We present and discuss approaches to solve SDALBP-1 without developing highly specialized procedures. Instead, we show how to use standard software and existing solution procedures for SALBP-1 in an effective manner.

4.1 Application of MIP standard software

Modelling is an important milestone in understanding a problem's structure and complexity. Furthermore, it always holds some capability for solving the problem directly.

In Sect. 3, we have formulated a nonlinear and two linear MIP. Because standard software for MIP (like, e.g., CPLEX and Xpress-MP) is not able to solve general nonlinear models in a direct and efficient manner, we focus on the linearized models.

Though relevant improvements have been obtained in accelerating such software packages, it has to be regarded that SDALBP is an NP-hard optimization problem. In case of SALBP, it has been shown that only rather small problem instances can be solved to optimality in a reasonable amount of time (cf., e.g., de Reyck and Herroelen 1997; Ugurdag et al. 1997; Pinnoi and Wilhelm 1997; Bockmayr and Pisaruk 2001). Because SDALBP is even more complex, the same is true for SDALBP and has been confirmed by preliminary computational tests. In order to improve the solution process, it is useful to include additional procedures to compute initial lower and upper bounds on the value of the objective function (number of stations m). This leads to great reductions in the number of variables and constraints thereby decreasing the size of the solution space to be examined.

Furthermore, bound computations can be based on solving relaxations even of a very complex model. Peeters and Degraeve (2006) present a Dantzig–Wolfe type formulation of SALBP-1, the LP-relaxation of which is solved using column generation combined with subgradient optimization. Computational experiments show that the resulting bound values are close to optimality. This approach could be transferred to our SDALBP-model.

4.2 Connections between SDALBP and SALBP

In order to solve SDALBP more efficiently, we describe an indirect approach which allows for applying the versatile arsenal of solution procedures available for SALBP. This approach is based on two interrelationships between SALBP and SDALBP, *relaxation and transformation*. In order to keep the presentation simple, we again focus on the type-1 problem versions. Transferring the approaches to the other versions is straightforward, because they are usually solved by search procedures based on SALBP-1 (cf. Scholl 1999, chaps. 4.2, 4.3).

Relaxation: generation of lower bounds on the number of stations

By completely ignoring sequence-dependent time increments (setting all sd_{ij} and sd_{ji} to zero), SDALBP reduces to a pure SALBP, i.e., SALBP is a relaxation of SDALBP. Therefore, any lower bound for SALBP is also a valid *lower bound* for SDALBP and can be used to judge the quality of heuristic solutions and to fathom nodes within a branch-and-bound tree. If the optimal solution to the relaxed instance is even feasible for the SDALBP-instance, i.e., the time increments can be realized within the idle times of the station loads built, this solution is also optimal for the SDALBP-instance. In order to examine feasibility, a scheduling problem has to be solved for each station load S_k that contains tasks which interact with each other. This problem is usually small and can be solved by enumeration easily.

Another relaxation of SDALBP is obtained by considering time increments while arbitrary task splitting among the stations is allowed. Then, the well-known capacity bound of SALBP (cf. Baybars 1986), i.e., $LM1 = \lceil t_{sum}/c \rceil$, is improved to $LM1' = \lceil t'_{sum}/c \rceil$.

Transformation: generation of upper bounds on the number of stations

By transforming *each* pair $(i,j) \in SD$ of task interactions into the directed precedence relation (i,j) or (j,i), the SDALBP-instance passes into a SALBP-instance. If the resulting precedence graph is acyclical and the (increased) time of neither task exceeds the cycle time, the corresponding SALBP-instance is *solvable*. Each feasible or even optimal solution to this SALBP-instance is also a feasible (but not necessarily optimal) solution to SDALBP. This feasibility is guaranteed because the task times contain the necessary time increments which is not true within the relaxed SALBP-instance. Thus, we get an *upper bound* for SDALBP.

To solve SDALBP to optimality, we have to construct *all* solvable transformed SALBP-instances systematically. The union of their feasible solution sets is equal to the feasible solution set of the original SDALBP-instance. Therefore, the optimal SDALBP-solution is obtained by taking from all optimal SALBP-solutions (the) one which requires a minimal number of stations. This implies that the SDALBP-instance is solvable if and only if there is at least one solvable transformed SALBP-instance.

The transformation of SDALBP to SALBP needs some further technical explanation: any SALBP-instance q is constructed by defining an individual precedence graph $G(q)$. Initially, it is set equal to the original precedence graph G ignoring the disjunctive arcs $(i,j) \in SD$. Each transformation of an interaction pair $(i,j) \in SD$ into a precedence relation (i,j) or (j,i) modifies $G(q)$. Without loss of generality, we consider the case of introducing the arc (i,j), because the reverse arc (j,i) can be handled identically by exchanging i and j in the following descriptions:

- The time of task j is increased to $t_j(q) := t_j(q) + sd_{ij}$. If j is part of several interaction pairs, its time is increased by each additional arc pointing to j independently such that the task time has to be modified more than once. Finally, we get the modified total work content $t_{sum}(q) = \sum_{j=1}^{n} t_j(q)$.

- Task i and its predecessors become predecessors of j, i.e., $P_j^*(q) := P_j^*(q) \cup \{i\} \cup P_i^*(q)$. Direct predecessors of j that are also predecessors of i are transformed into indirect predecessors of j, i.e., $P_j(q) := P_j(q) \cup \{i\} - P_i^*(q)$. Analogously, task j and its successors become successors of i, i.e., $F_i^*(q) := F_i^*(q) \cup \{j\} \cup F_j^*(q)$. Direct successors of i that are also followers of j become indirect successors of i, i.e., $F_i(q) := F_i(q) \cup \{j\} - F_j^*(q)$.[3]
- If there is another task interaction $(h,k) \in$ SD or $(k,h) \in$ SD with $h \in P_i^*(q) \cup \{i\}$ and $k \in F_j^*(q) \cup \{j\}$, it is necessary to introduce the (then redundant) arc (h,k) instead of (k,h) in order to avoid an infeasible (cyclical) precedence graph.
- In case of adding arcs (j,i) with $j > i$, the task numbering is not topological anymore and has to be modified if required by the solution procedure.

Because each interaction pair can be transformed into two reverse arcs, we get up to $2^{|\mathrm{SD}|}$ feasible combinations of arc orientations, i.e., $Q \leq 2^{|\mathrm{SD}|}$ solvable SALBP-instances $q = 1, \ldots, Q$. In at least one of those instances the directions of the interaction pairs coincide with the order in which the tasks are assigned to the stations in an optimal solution for SDALBP. Therefore, the minimum number m^* of stations for the SDALBP-instance is equal to the minimum of the numbers of stations $m^*(q)$ required by all SALBP-instances, i.e., $m^* = min\{m^*(q) | q = 1, Q\}$.

Figure 5 shows and relates the solution spaces of SDALBP, the relaxed SALBP, and the transformed SALBPs. By ignoring all disjunctive precedence constraints in the relaxed SALBP, the solution space is enlarged because it additionally contains SDALBP-infeasible SALBP-solutions caused by cycles in the precedence constraints and/or violated cycle time constraints. The transformed SALBP-instances SI_q obtained by systematically combining directed disjunctive arcs are solvable (instances SI_1 to SI_Q) or not (SI_{Q+1} to $\mathrm{SI}_{Q'}$ with $Q' = 2^{|\mathrm{SD}|}$). The solution spaces of the solvable instances completely cover the solution space of SDALBP, the unsolvable ones only fall into the solution space of the relaxed SALBP but do not cover it completely.

Example We consider the SDALBP-instance with the precedence graph and interaction pairs of Fig. 2 as well as cycle time $c = 10$. The *relaxed* SALBP-instance is obtained by removing all disjunctive arcs. When constructing all solvable *transformed* SALBP-instances, we find $2^4 = 16$ combinations of directing all interaction pairs. From this set of possible instances, only $Q = 10$ turn out to be solvable, whereas the remaining have cycles in their precedence graphs. Figure 6 shows the solvable instances. The arcs resulting from orientating the pairs are drawn as bold lines, redundant (original or additional) arcs are drawn as dashed lines. The task times are modified according to the rules given above.

In the literature, there exist a lot of bounding arguments and exact solution procedures for SALBP-1. In order not to overload the paper and not to repeat

[3] Notice that this implies indirect precedence relations between the predecessors of i and the successors of j.

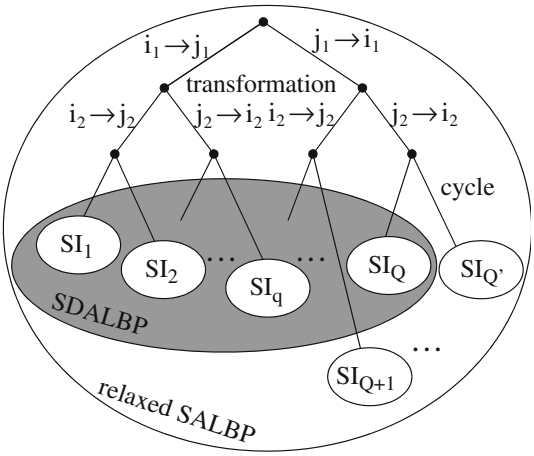

Fig. 5 Relaxation and transformation

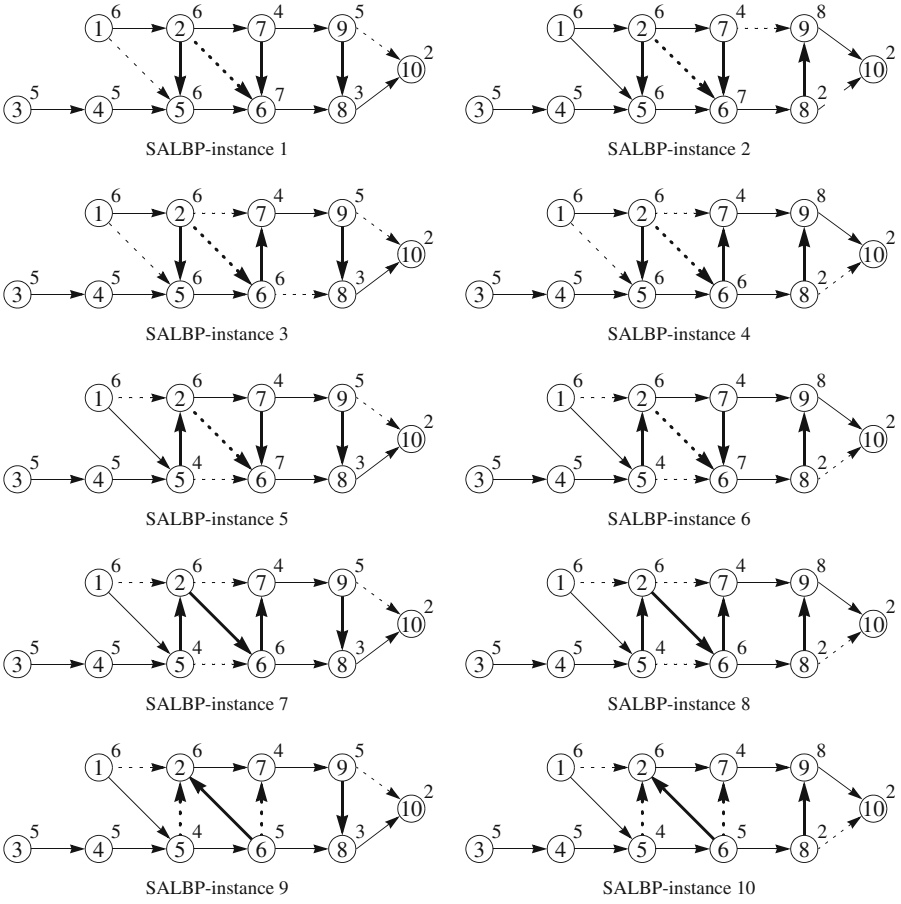

Fig. 6 Set of SALBP-instances representing the SDALBP-instance of Fig. 2

well-known facts, we do without explaining such procedures but refer to the surveys of Baybars (1986), Ghosh and Gagnon (1989), Scholl (1999, chap. 2.2.2, 4.1) as well as Scholl and Becker (2006). Computational experiments show that the branch-and-bound procedure SALOME of Scholl and Klein (1997) is supposably the best exact solution method which is available for SALBP-1 [cf. the computational experiments in Scholl and Klein (1997, 1999), Sprecher (1999)]. Therefore, we propose to apply SALOME (including its arsenal of bounding, reduction and dominance rules) within the framework defined above. For detailed descriptions of SALOME, see Scholl and Klein (1997) as well as Scholl (1999, chap. 4.1.4).

4.3 Basic search procedure for SDALBP

A basic procedure for SDALBP utilizes the above-mentioned approaches intuitively:

1. Solving the relaxed SALBP-instance, which we denominate as instance 0, results in a minimal number of stations $m^*(0)$. If this solution is feasible for SDALBP, i.e., if there exists a sequence of tasks such that the cycle time is not exceeded for each station, the optimal solution is found and the procedure stops with $m^* := m^*(0)$.
   Otherwise, we get a *global lower bound* $\mathrm{LB} = \max\{m^*(0), \mathrm{LM1}'\}$, for the number of stations required for SDALBP. As initial *global upper bound*, we use $\mathrm{UB} := n + 1$ in order not to exclude the optimal solution which might have up to n stations when all tasks require an individual one.
2. The transformed SALBP-instances $q = 1, \ldots, Q$ are generated and solved one after the other. Each time, $m^*(q) < \mathrm{UB}$ is true, the optimal solution for instance q is better than the incumbent solution to SDALBP. Then, we set $\mathrm{UB} := m^*(q)$ and store the new solution as incumbent one. If $\mathrm{UB} = \mathrm{LB}$ or $q = Q$, the procedure stops with $m^* := \mathrm{UB}$.

When applying SALOME to an instance q in step 2, the upper bound $\mathrm{UB}(q)$ is initialized to the value of the global upper bound UB. Thus, SALOME fathoms any node of the branch-and-bound tree which cannot lead to an improved SDALBP-solution even if the optimal SALBP-solution of the instance q is missed. This avoids unnecessary computations and takes place whenever $m^*(q) \geq \mathrm{UB}$ holds.

Example By solving the instance 0 with SALOME in step 1, we may get the optimal solution $(\{3,4\},\{1,5\},\{2,7\},\{6,9\},\{8,10\})$ with $m^*(0) = 5$ stations. Adapting this solution to SDALBP shows that it is not feasible, because the fourth station gets overloaded with $t(S_4) = (t_6 + \mathrm{sd}_{26} + \mathrm{sd}_{76}) + t_9 = 12$. Instead, we get the lower bound $\mathrm{LB} = \max\{5, \lceil 45/10 \rceil\} = 5$. In step 2, the instances q = 1,…, 10 are successively solved by SALOME until the termination condition UB = LB is fulfilled or all instances have been examined. Table 3 shows the minimal numbers of stations m*(q) which would be obtained by SALOME if necessary.

Table 3 Optima for solvable SALBP-instances

q	1	2	3	4	5	6	7	8	9	10
$m^*(q)$	6	6	6	7	5	5	5	5	5	5

In iteration $q = 1$, we get UB $:= m^*(1) = 6$. The next improvement to UB $:= m^*(5) = 5$ is attained in iteration $q = 5$. Then, the procedures stops due to LB = UB = 5. The (unique) optimal solution to instance 5 which is $(\{3,4\},\{1,5\},\{2,7\},\{6\},\{9,8,10\})$ constitutes (one of) the optimal SDALBP-solution(s) with $m^* = 5$ stations. In total, we must solve only six SALBP-instances to optimality in order to solve the SDALBP-instance.

Notice that the procedure would have stopped with the optimal SDALBP-solution immediately, if we had found the alternative optimal solution $(\{3,4\}, \{1,5\},\{2,7\},\{6,8\},\{9,10\})$ to instance 0 in step 1.

Remark If the computation time required is available, the basic procedure finds an optimal solution to SDALBP with m^* stations. However, if the procedure has stopped due to a time limit, it has only found a feasible (incumbent) solution with UB stations provided that a solvable instance has been generated within the time limit. Furthermore, a lower bound LB is known such that the solution quality can be judged.

4.4 Advanced search procedure for SDALBP

Obviously, the basic procedure described above potentially requires excessive computations, because it is not always possible to find an optimal solution in step 1 and the number of instances to be solved in step 2 grows with increasing number of interaction pairs exponentially. This is particularly relevant, because each of those instances is NP-hard by itself.

In order to reduce the number of SALBP-instances to be solved, we enhance the solution procedure by computing and utilizing lower and upper bounds for each SALBP-instance $q = 1,\ldots,Q$ (prior to solving any NP-hard instance to optimality) and using some sorting mechanism such that promising instances are considered first.

A lower bound $\mathrm{LB}(q) \leq m^*(q)$ is defined by the maximal value obtained for any lower bound argument applied to the SALBP-instance q. An upper bound $UB(q) \geq m^*(q)$ is given by the number of stations required in the best feasible solution for the SALBP-instance q that is obtained by any heuristic procedure applied to q.

An *global lower bound* LB for the original SDALBP-instance is given by the minimum of the lower bounds on all transformed SALBP-instances, i.e., $\mathrm{LB} = \min\{\mathrm{LB}(q) | q = 1,\ldots,Q\}$. In each instance q, we have $t_{\mathrm{sum}}(q) \geq t'_{\mathrm{sum}}$ and, thus, $\mathrm{LB} \geq \mathrm{LM1}(q) \geq \mathrm{LM1}'$.

As *global upper bound* for SDALBP, we use the smallest upper bound for any transformed SALBP-instance, i.e., UB := min {UB(q) | $q = 1, \ldots, Q$}. If such an upper bound is not (yet) available for any instance q, we may set UB(q) := $n + 1$.

The *advanced procedure* utilizes the above-mentioned principles consequently and consists of the following steps ('STOP' immediately terminates the whole procedure):

1. Initialize the search by performing the operations (a) to (c):
   (a) Compute a lower bound LB(0) for the relaxed instance 0 and initialize the global lower bound by setting LB := max {LB(0), LM1′}.
   (b) Initialize the global upper bound by setting UB := $n+1$. Additionally, apply a computationally inexpensive heuristic to find a feasible SALBP-solution for instance 0 with upper bound UB(0). If this solution is also feasible for SDALBP, set UB := UB(0) and store the solution as incumbent solution. If UB = LB, then STOP.
   (c) Apply a computationally inexpensive heuristic to find a feasible SDALBP-solution, which provides a (global) upper bound UB, and store it as incumbent solution. If UB = LB, then STOP.
   (d) Initialize the list L of unsolved instances to be an empty list.
2. Generate the solvable instances $q = 1, \ldots, Q$ in a systematic (lexicographic) manner. For each interaction pair $(i, j) \in$ SD the orientation with shorter time increment is considered first such that instance 1 includes the arc (i, j) if $sd_{ij} \leq sd_{ji}$ and the reverse arc (j, i) else. As a consequence, the total task time of instance 1 is equal to t'_{sum} and the lower bound LM1′ is minimal.
   For each instance q, the following operations are performed in the given order such that only necessary computations are made and the list is kept as small as possible:
   (a) Compute a lower bound LB(q) by applying any promising bound argument for SALBP (cf. Scholl and Klein 1997, 1999) and taking the maximum of the resulting values.
   (b) If $LB(q) \geq$ UB, the instance q cannot contribute to improving the incumbent solution and is discarded (fathomed). Thus, skip the remaining operations (c) to (e).
   (c) Apply some computationally inexpensive heuristic to compute a feasible solution that provides an upper bound UB(q).
   (d) If UB(q) < UB, we have found an improved SDALBP-solution. Set UB := UB(q) and store the corresponding solution as the incumbent one. If UB = LB, then STOP.
   (e) If UB(q) = LB(q), the instance q is solved, i.e. $m^*(q)$ = UB(q), and, thus, discarded. Otherwise, the instance q is stored in L, together with the interval [LB(q), UB(q)].
3. At the end of step 2, the list L contains a number of still unsolved SALBP-instances. Perform the following operations to adjust and order the list:
   (a) According to step 2b, discard all instances q from L for which LB(q) $\geq$ UB is now fulfilled. A number $R \leq Q$ of promising instances remains. If $R = 0$, then STOP.

(b) Consider L as a list of the remaining instance numbers, i.e. $L = < h_1, \ldots, h_r, \ldots, h_R]$, and sort its entries such that $\mathrm{LB}(h_r) \leq \mathrm{LB}(h_{r+1})$ holds for all $r = 1, \ldots, R-1$. If several instances have the same bound value, sort them according to nondecreasing values of the total work content $t_{\mathrm{sum}}(h_r)$ with second-level ties broken in the order of initial instance numbers. Set $\mathrm{LB} := \mathrm{LB}(h_1) = \min\{\mathrm{LB}(h_r) \mid r = 1, \ldots, R\}$.

4. Consider the instances h_r in their sorting order $r = 1, \ldots, R$ and perform the following operations, respectively:
   (a) Solve the instance h_r by applying SALOME starting from the lower bound $\mathrm{LB}(h_r)$ and the upper bound $\mathrm{UB}(h_r) := \mathrm{UB}$. This avoids unnecessary computations and takes place whenever $m^*(h_r) \geq \mathrm{UB}$ holds. Otherwise, SALOME finds an improved incumbent solution to SDALBP with $\mathrm{UB} := m^*(h_r)$ stations which has to be stored. If $\mathrm{UB} = \mathrm{LB}$, then STOP.
   (b) If the last instance $r = R$ has been solved, then STOP. If $\mathrm{LB}(h_{r+1}) > \mathrm{LB}$, then set $\mathrm{LB} := \mathrm{LB}(h_{r+1})$, because all instances which might have a solution with a smaller number of stations have already been examined. If $\mathrm{UB} = \mathrm{LB}$, then STOP.

Result If the procedure has terminated by the command 'STOP', it has found an optimal solution to SDALBP with $m^* := \mathrm{UB}$ and the task assignments stored as incumbent solution. If the procedure has stopped due to a time limit, it has only found a feasible (incumbent) solution with UB stations and a lower bound LB.

Explanations and details:

- The advanced search procedure follows the principle of a *lower bound search* which is successfully used to solve SALBP-2 by iteratively solving instances of SALBP-1 in order of increasing cycle times (cf. Hackman et al. 1989; Scholl 1999, chap. 4.2.2).
- For determining an initial upper bound in the steps 1b, 1c and 2b, simple heuristic approaches such as priority rule based procedures (see, e.g., Talbot et al. 1986; Hackman et al. 1989; Scholl and Voß 1996) are primarily recommendable because they find solutions very quickly.
  In step 1c, we apply a station-oriented procedure in forward direction using the priority list built by sorting the tasks in nondecreasing order of task times with ties being broken in favour of tasks with most direct successors (cf. Johnson 1988). In order to find a feasible solution, each task i which would lead to $t_j + \mathrm{sd}_{ij} > c$ for an unassigned interacting task $j \in \mathrm{SD}_i$ is delayed until j has been assigned.
  In steps 1b and 2b, we utilize the ability of SALOME to find good feasible solutions quickly. SALOME is applied to the respective SALBP-instance and immediately stopped when a first feasible solution is retrieved. In fact, this is a combination of the above-mentioned priority rule with the local lower bound method which directs the search of SALOME to promising parts of the search tree first (cf. Scholl and Klein 1997, 1999).

Table 4 Lower bounds of solvable SALBP-instances

q	0	1	2	3	4	5	6	7	8	9	10
$t_{sum}(q)$	44	49	51	48	50	47	49	46	48	45	47
LM1(q)	5	5	6	5	5	5	5	5	5	5	5
LM2(q)	4	6	6	6	6	5	5	5	5	4	5
LB(q)	5	6	6	6	6	5	5	5	5	5	5
UB(q)	5	6	6	6	7	5	5	5	6	5	5

- If computer memory is a very scarce resource, the step 3a might be performed always after finding an improved incumbent in step 2d alternatively.
- When setting LB at the end of step 3b, we need not take into account its old value determined in step 1a provided that the same set of bound arguments is computed for each SALBP-instance considered. Since instance 0 relaxes any transformed instance $q = 1, \ldots, Q$ by ignoring the additional precedence relations and the induced time increments, $LB(q) \geq LB(0)$ holds for each q. Furthermore, $LB(q) \geq LM1'$ is true for each q, because no instance realizes smaller time increments than are considered when computing LM1′. Thus, the new value of LB is at least as great as the old one.

Example To keep our computations simple, we assume that only the most elementary lower bound arguments are applied (cf. Johnson 1988). Besides LM1(q), only the counting bound LM2(q) which ignores all tasks with $t_j(q) < c/2$ is used. Because only one task $t_j(q) > c/2$ and at most two tasks with $t_j(q) = c/2$ can be assigned to each station, LM2 counts (and rounds up) those tasks (given a weight of 1 to the first and 1/2 to the second group of tasks).

Table 4 shows the lower bound values with LB(q) being the maximum of LM1(q) and LM2(q) including those which need not be computed by the search procedure. Furthermore, Table 4 contains the upper bounds UB(q) obtained by stopping SALOME immediately after finding the first feasible solution as described above.

The instances have to be sorted (and renumbered) in the order $< 9, 7, 5, 10, 8, 6|3, 1, 4, 2]$ and we get $LB = LB(9) = 5$. The first six instances have a lower bound of five and the remaining ones a lower bound of six stations. Provided that the solution found for instance 0 is not feasible (see Sect. 4.3), the procedure starts with heuristically solving the instance 9. As the first solution $(\{3, 4\}, \{1\}, \{5, 6\}, \{2, 7\}, \{9, 8, 10\})$ found within SALOME fulfils $UB(9) = LB(9) = 5$, it is an optimal solution to instance 9. Furthermore, it is an optimal solution to the original SDALBP-instance due to $UB := UB(9) = 5 = LB$.

Note that it is not necessary to compute any other value UB(q) provided in Table 4, because the search stops immediately after determining UB(9).

5 Computational experiments

In order to examine the performance of the solution procedures described in Sect. 4, we perform computational experiments based on systematically varied benchmark test instances. In particular, the following questions should be answered:

- How do the number and extents of task interactions influence the problem's complexity?
- Up to which problem size is it sufficient to use standard MIP-software to solve SDALBP-instances? Is the obvious expectation justified that solving the (simplified) model L2 is easier than solving L1?
- How do the search procedures perform when applied to different problem classes. Are the refinements contained in the advanced procedure successful in reducing the computing times? Is it necessary to develop specialized procedures for SDALBP or is it sufficient to apply SALBP-based search procedures?
- As a model base for a real-world line balancing problem, it is possible to use either SALBP or SDALBP. In this context, we are interested in the negative influence of ignoring possible task time increments and/or modeling them as unique precedence relations. In which cases is it necessary to explicitly model those interactions (use SDALBP) and how should these interactions be handled if SALBP is chosen to be the model base?

5.1 Test data generation

As a basis for generating test instances for SDALBP-1 we use the well-known data sets for SALBP-1. We combine the three original data sets from Talbot et al. (1986), Hoffmann (1992) and Scholl (1993) to a single set with 269 instances. The instances are based on 25 precedence graphs having 8–297 nodes each connected to several cycle times. For a detailed description of the data sets and the characteristics of the underlying precedence graphs see Scholl (1999, chap. 7.1) and the "assembly line balancing research homepage" http://www.assembly-line-balancing.de, where the corresponding data files can be downloaded from.

Two parameters are used to systematically complement the SALBP-instances with sequence-dependent task time increments:

- sr defines the number of interacting task pairs as a portion of the number of tasks, i.e., $|\mathrm{SD}| = \lceil \mathrm{sr} \times n \rceil$
- ir maximal relative increment of the task time t_j by a task interaction, i.e., $\mathrm{sd}_{ij} \leq \mathrm{ir} \times t_j$

In Sect. 2, we have explained that actual interactions between tasks i and j are considered in two ways when the real problem is to be modelled as a SALBP-instance:

(a) A precedence relation (i,j) is introduced which represents the (locally) favourable ordering of i and j. The task time t_j, thus, includes the time increment sd_{ij}.
(b) The task interaction is ignored thereby assuming that the task times rely on unhindered standard procedures. As a consequence, the task time of the later task is underestimated.

We generate SDALBP-instances which reflect these reductions possibly made when modelling the real problem as an SALBP-instance by performing the following steps based on the original topological task numbering:

1. Determine the set SD by randomly selecting a total number of $|\mathrm{SD}| = \lceil \mathrm{sr} \times \mathrm{n} \rceil$ task pairs such that $(i,j) \in E$ (case a) or $(i,j) \notin E^*$ with $i < j$ (case b) is fulfilled. The selection probability is identical for each such pair irrespective of its type (a or b).
2. Determine task time increments (and adapted task times) for all pairs of SD corresponding to the cases explained above:
   (a) For all task pairs $(i,j) \in \mathrm{SD} \cap E$: Because the original task time t_j already includes the time increment sd_{ij}, the pure task time is given by $t'_j = t_j - \mathrm{sd}_{ij}$. Therefore, sd_{ij} is randomly chosen from the interval $\left[0, \frac{\mathrm{ir}}{1+\mathrm{ir}} \times t_j\right]$ and t'_j is set to $t_j - \mathrm{sd}_{ij}$.
   The increment sd_{ji} is randomly chosen from $[0, \mathrm{ir} \times t]$ and t_i needs no modification, because the original t_i was not affected by the task j. Due to the assumption that the relation (i,j) was introduced as the favourable ordering, i.e., $\mathrm{sd}_{ij} \leq \mathrm{sd}_{ji}$, we finally set $\mathrm{sd}_{ji} := \max\{\mathrm{sd}_{ij}, \mathrm{sd}_{ji}\}$.
   (b) For all task pairs $(i,j) \in \mathrm{SD} - E$: Neither task time already includes a time increment. Therefore, we randomly choose sd_{ij} from $[0, \mathrm{ir} \times t_j]$ and sd_{ji} from $[0, \mathrm{ir} \times t_i]$, while the task times t_i and t_j do not need a modification.

In order to avoid that some ordering (i,j) or (j,i) is infeasible at the outset, it is additionally ensured that $t_j + \mathrm{sd}_{ij} \leq c$ and $t_i + \mathrm{sd}_{ji} \leq c$ are fulfilled by restricting the used intervals correspondingly.

By independently setting the parameters (sr and ir) to four values each, we get 16 SDALBP-data sets with a total of 4,304 instances):

- sr = 0.02, 0.05, 0.10, 0.15 (a few to a considerable number of interaction pairs)
- ir = 0.25, 0.50, 0.75, 1.0 (small to considerable task time increments; notice that the average increment is smaller than ir $\times t_j/2$ due to the construction rules).

5.2 Solving the model by standard software

The linear models L1 and L2 developed in Sect. 3 were implemented using the XPress-MP software of Dash Associates (http://www.dashoptimization.com).

Table 5 Results for solving the models L1 and L2 by XPress-MP

		Model L1				Model L2			
sr	ir	# Opt.	# Found	Rel. dev.(%)	Total time	# Opt.	# Found	Rel. dev.(%)	Total time
0.02	0.25	110	158	2.51	322.04	100	158	2.55	335.15
	0.50	109	158	2.52	322.82	97	158	2.56	338.26
	0.75	109	157	2.57	322.80	96	157	2.61	341.05
	1.00	109	156	2.61	323.63	101	158	2.55	332.79
0.05	0.25	115	166	2.35	315.53	94	165	2.40	344.40
	0.50	113	165	2.35	318.83	100	165	2.39	330.49
	0.75	114	163	2.42	315.19	99	163	2.46	333.49
	1.00	112	163	2.45	317.49	102	165	2.40	327.28
0.10	0.25	110	162	2.40	312.99	94	164	2.32	345.12
	0.50	105	163	2.41	320.58	90	165	2.29	350.28
	0.75	108	155	2.58	317.10	93	156	2.52	347.32
	1.00	109	155	2.63	317.25	94	156	2.59	339.22
0.15	0.25	111	171	2.11	318.62	88	168	2.24	354.54
	0.50	110	158	2.42	335.70	81	156	2.52	369.59
	0.75	112	157	2.55	330.93	85	153	2.75	360.85
	1.00	106	149	2.81	336.84	82	149	2.84	366.61
Total		1,762	2,556	2.48	321.77	1,496	2,556	2.50	344.78

In order to reduce the number of variables and to accelerate the solution process, the global lower bound on the number of stations is set to LM1′.

To compute an initial upper bound, we use the same simple priority rule-based procedure as applied in step 1b of the advanced search procedure (forward pass, station-oriented assignment, first-level priority: maximum task time, second-level priority: maximum number of immediate followers; cf. Sect. 4.4). All computations and the model itself were coded by means of the Mosel programming language. The experiments were run on a personal computer with an Intel Pentium IV processor of 3 GHz clock speed and 512 MB of RAM. For each instance a time limit of 500 s was imposed. The difference in overall performance, measured in relative deviations from optimality, was further tested for statistical significance using the Wilcoxon Signed Rank Test (see Appendix).

Table 5 summarizes the results based on the following measures:

- # Opt.: number of proven optima found (out of 269 instances)
- # Found: number of optima found (the optimal solution is retrieved, i.e., the final UB is equal to m^*, but could not necessarily be proven within the given time span)
- Rel. dev.: average relative deviation from minimal number of stations (or best known lower bound if the optimum is still unknown)
- Total time: average computation time (contains a value of 500 s for time out cases)

The last row contains aggregate information on all data sets.

Table 5 reveals that model L1 is able to solve about 40% of the instances to optimality, for about further 20% of the instances the optimal solution is found but cannot be proven within the time span available. The model L2 solves only

Table 6 Results of both search procedure with regard to problem size

		Model L1				Model L2			
# Tasks	# Inst.	# Opt.	# Found	Rel.dev.(%)	Total time	# Opt.	# Found	Rel. dev.(%)	Total time
≤ 30	880	826	858	0.29	43.97	820	853	0.40	51.35
31–74	960	518	574	3.44	259.14	517	578	3.45	275.87
75–110	1,216	373	675	2.56	399.59	353	676	2.55	455.09
> 110	1,248	449	449	3.21	490.01	449	449	3.20	497.20

about 35% of the instances to optimality but finds (by chance) the same number of optimal solutions. Therefore, the difference in relative deviations is rather small and statistically insignificant (see Appendix).

The similar behaviour with respect to "# Found" is primarily due to the initial heuristic which finds the optimal number of stations m^* for 2,148 instances (the heuristic by itself produces an average relative deviation from optimality of 4.02%). Then, the models have only to determine an assignment of the tasks to this (or a lower) number of stations. Nevertheless, both models fail for many instances as documented by the difference of "# Found" and "# Opt." with L2 still being slightly worse than L1. Thus, we find the unexpected result, that avoiding the large number of transitivity constraints in model L2 does not have a positive effect on the model's solution capabilities. By the way of contrast, the polytope defined by those constraints obviously provides stronger LP-relaxations which are solved to compute lower bounds within the branch and bound procedure of XPress-MP such that L1 is able to prove more instances to optimality, especially when the number of interaction pairs is high (sr=0.10, 0.15).

In order to find out, up to which problem size it is recommendable to apply either model, we aggregate the 4,304 problem instances into four groups with regard to the number of tasks considered. Table 6 displays the results using the following additional notation:

- # Tasks: range of the number of tasks in the respective group
- # Inst.: total number of problem instances in this group

Generally, the application of either model can only be recommended to solve problem instances with up to 30 tasks. Larger instances can be solved to optimality only in rather simple cases where the initial heuristic succeeds in finding the optimal number of stations.

5.3 Applying the search procedures

The search procedures developed in Sect. 4 have been implemented using Borland Delphi 7.0 based on the code of SALOME already used in Scholl and Klein (1997, 1999).

Table 7 Results for basic procedure

sr	ir	# Opt.	# Found	Rel. dev.(%)	Total time	Time for incumb.	# Generated	# Solvable	# Stored	# Solved
0.02	0.25	256	256	0.12	28.30	16.55	1.5	1.5	1.0	1.2
	0.50	255	255	0.13	28.45	15.03	1.4	1.4	1.0	1.3
	0.75	254	254	0.15	29.02	15.58	1.4	1.4	1.0	1.3
	1.00	255	255	0.12	28.33	16.58	1.4	1.4	1.0	1.3
0.05	0.25	255	255	0.14	27.20	14.62	4.3	3.2	1.0	1.8
	0.50	256	257	0.11	27.73	18.43	4.3	2.9	1.0	1.9
	0.75	254	254	0.15	29.55	15.75	26.7	19.7	1.0	1.8
	1.00	255	255	0.14	26.98	16.85	6.2	3.7	1.0	1.8
0.10	0.25	257	257	0.12	24.66	17.68	227.5	93.0	1.0	4.0
	0.50	257	257	0.13	24.95	13.96	220.3	79.9	1.0	4.5
	0.75	260	260	0.10	18.98	11.12	206.7	91.9	1.0	4.0
	1.00	258	258	0.13	22.93	14.34	279.1	139.9	1.0	3.9
0.15	0.25	252	252	1.25	29.73	11.89	3,600.8	582.5	1.0	26.9
	0.50	253	253	0.88	29.96	16.03	2,402.3	251.7	1.0	15.6
	0.75	254	254	0.50	28.35	18.62	2,356.3	318.0	1.0	14.4
	1.00	255	255	0.84	27.07	16.40	1,739.4	382.5	1.0	17.0
Total		4,086	4,087	0.31	27.01	15.59	692.5	123.4	1.0	6.4

Table 8 Results for advanced procedure

sr	ir	# Opt.	# Found	Rel. dev.(%)	Total time	Time for incumb.	# Generated	# Solvable	# Stored	# Solved
0.02	0.25	257	257	0.11	25.28	1.94	3.3	3.0	1.6	1.4
	0.50	257	257	0.11	24.88	1.56	3.4	3.0	1.6	1.4
	0.75	257	257	0.11	24.57	1.24	3.1	2.8	1.6	1.4
	1.00	257	257	0.11	24.61	1.27	3.2	2.9	1.6	1.4
0.05	0.25	257	257	0.13	23.97	0.84	33.9	28.1	7.4	2.0
	0.50	260	260	0.08	18.16	0.61	10.3	8.2	5.6	1.8
	0.75	260	260	0.10	18.13	0.60	10.1	8.2	5.3	1.8
	1.00	260	260	0.10	18.10	0.57	9.8	7.7	4.9	1.7
0.10	0.25	263	263	0.06	15.54	1.94	336.3	71.4	52.3	7.1
	0.50	261	261	0.08	19.72	3.23	567.0	139.1	59.2	6.6
	0.75	260	260	0.09	20.45	2.32	687.1	189.0	60.4	5.7
	1.00	261	261	0.08	19.34	3.00	698.5	195.2	56.6	5.6
0.15	0.25	260	260	0.08	22.39	1.72	2,546.4	262.7	49.3	36.1
	0.50	258	258	0.11	25.88	1.70	3,098.5	409.0	54.9	24.8
	0.75	261	261	0.07	18.99	1.79	2,184.8	238.1	43.0	23.0
	1.00	262	262	0.06	17.00	1.54	2,009.0	265.2	48.4	25.0
Total		4,151	4,151	0.09	21.06	1.62	762.8	114.6	28.3	9.2

Again, a time limit of 500 s was imposed for each instance. Additionally, the computation time was restricted to at most 50 s for step 1 of the basic procedure in order not to get stuck in a useless optimization of a relaxed problem.

Tables 7 and Table 8 show the results obtained for the basic and the advanced search procedure, respectively. The following measures have been used additionally:

- Time for incumb.: average time to find the best solution (or optimum, if no time out occurs)
- # Generated: average number of SALBP-instances generated
- # Solvable: average number of solvable SALBP-instances generated
- # Stored: average number of SALBP-instances stored
- # Solved: average number of SALBP-instances solved to optimality.

The average results (contained in the last rows of the tables) can be summarized as follows:

- The basic procedure solves 4,086 of the 4,304 instances to optimality (94.9%), whereas the advanced procedure finds 4,151 proven optima (96.4%). The average relative deviation from optimality is 0.31 and 0.09%, respectively, so that it can be stated that the advanced procedure performs better than the basic procedure which in turn outperforms XPress-MP (with all differences being statistically significant, see Appendix).
- The average computation times are 27.01 and 21.06 s per instance. A part of the time reduction by the advanced procedure is due to the larger number of solved instances. Another part is due to finding useful SALBP-instances, which are easily solved and provide good bounds for SDALBP, earlier (preprocessing in step 2 and sorting in step 3).
- Considering the time to find the optimal or best-known solution shows a much greater time reduction. While the basic procedure requires an average time of 15.59 s, the advanced procedure finds this solution already after 1.62 s on average. This is particularly useful in solving large real-world instances, where a good heuristic solution is searched for in a small time span.
- Both procedures have to generate about 700 SALBP-instances per SDALBP-instance. In both cases less than 20% of them are solvable. Because the basic procedure solves each instance immediately after generating it, no further instances must be stored in a candidate list. The advanced procedure is able to eliminate about 75% of the solvable instances by the bound computations in step 2, only the remaining ones need to be stored.
  In both cases, only a small part of the solvable instances (5.2 vs. 8.0%) have actually to be solved completely, because mostly it is possible to terminate SALOME before the SALBP-optimum has been found due to the upper bound already known. The smaller number obtained by the basic procedure is due to the fact that it finds a solution of the relaxed SALBP-instance 0 in step 1 which is optimal for SDALBP for 1,844 (42.8%) instances.

Considering the variation of the problem parameters sr and ir, the following can be stated:

- As expected, the number of instances to be solved significantly increases with the relative number sr of interaction pairs. Nevertheless, the computation times do not increase considerably. By the way of contrast, they are even reduced in some cases. It can be assumed that this is due to the reduction of the order strength by transforming more precedence relations into interaction pairs thereby allowing for better utilization of the station time.

Table 9 Results of both search procedures with regard to problem size

		Basic procedure				Advanced procedure			
# Tasks	# Inst.	# Opt.	# Found	Rel. dev.(%)	Total time	# Opt.	# Found	Rel. dev.(%)	Total time
≤ 30	880	880	880	0.00	0.02	880	880	0.00	0.01
31-74	960	960	960	0.00	1.14	960	960	0.00	5.51
75-110	1,216	1,160	1,161	0.16	24.99	1,169	1,169	0.13	21.58
> 110	1,248	1,086	1,086	0.92	68.16	1,142	1,142	0.19	47.37

- In the case of many interaction pairs (sr $= 0.15$), the basic procedure is unable to find a good feasible solution in some cases, because most instances are not solvable. Thus, the average relative deviation from optimality is much greater than with fewer interaction pairs. This disadvantage is overcome by the heuristic applied in step 1b of the advanced procedure.
- The maximal increment factor ir has no remarkable influence on solution quality and computation times.

In order to investigate the influence of the problem size on the performance of both search procedures, we consider the respectively classified results in Table 9.

The results reveal that both procedures solve all smaller instances with less than 75 tasks to optimality very quickly, with the basic procedure being slightly superior with regard to solution times. For larger instances, the effort of intensified pruning results in an improved performance and the advanced procedure clearly outperforms the basic search. It is further remarkable that the advanced procedure is able to solve and prove to optimality 96% of the larger instances (75–110 tasks) and 92% of the largest instances in the test bed including problems up to 297 tasks.

5.4 Comparing SALBP- and SDALBP-solutions

We have generated the SDALBP-instances in such a manner that the corresponding SALBP-instances could have been derived from the more realistic SDALBP-instances by ignoring all task interactions (cf. the two ways of transformation described in Sect. 5.1). Thus, we are able to simulate the loss in precision by using SALBP instead of SDALBP for our test data set. If it is assumed that practitioners will need to approximate in a similar fashion in order to comply with the standard SALBP-formulation, our results might provide further indications on how real-world problems might be affected by modelling decisions.

We distinguish three cases in order to assess the differences between optimal SALBP- and their corresponding SDALBP-solutions:

1. The optimal SALBP-solution can be SDALBP-infeasible. This will be the case whenever the total time increment of ignored interactions of the tasks assigned to a station exceeds the station's idle time. In practice this will

Table 10 SALBP versus SDALBP

sr	ir	Infeas(%)	Under(%)	Over(%)
0.02	0.25	20.45	3.72	2.60
	0.50	39.03	4.83	2.23
	0.75	55.39	5.58	1.86
	1.00	68.40	5.20	1.86
0.05	0.25	23.05	10.41	3.35
	0.50	43.12	11.52	3.72
	0.75	57.99	11.90	3.35
	1.00	69.89	11.90	2.97
0.10	0.25	23.42	16.73	4.83
	0.50	44.24	17.10	3.72
	0.75	58.36	17.84	3.35
	1.00	72.86	17.84	3.35
0.15	0.25	23.79	21.56	3.35
	0.50	44.61	21.93	3.35
	0.75	59.11	22.30	2.60
	1.00	73.61	22.30	2.60
Total		48.58	13.92	3.07

result to overloads at stations which will need to be compensated by costly counteractions or a reassignment of tasks.

2. The optimal number of stations of a SALBP-instance is lower than in the optimal SDALBP-solution. As a consequence the whole assembly line will be dimensioned inappropriately, so that even a reassignment of tasks will not yield a feasible solution. Obviously, all SALBP-solutions in this category are also SDALBP-infeasible.
3. The optimal number of stations of a SALBP-instance is greater than in the optimal SDALBP-solution. In this case, the capacity requirements are overestimated, so that investment costs could have been saved. Note, that SALBP-solutions in this category can nevertheless be SDALBP-infeasible.

The results summarized in Table 10 show that almost half of all SALBP-solutions are in fact SDALBP-infeasible (column "Infeas"). The portion of infeasible solutions drastically increases from about 20 to 70% when the maximal relative time increment ir is increased from 0.25 to 1.0. That is, in case of considerable task time increments, it is not meaningful to ignore these increments by modeling and solving the problem as a SALBP.

A comparatively high portion additionally underestimates the total number of stations (column "Under"), which would lead to even more critical adjustments in practice. The number of underestimations significantly increases the more interaction pairs are considered.

Opposed to that, comparatively few overestimations can be observed (column "Over"). The number of overestimations appears to be rather stable and independent of the two parameters.

In general, the underestimation of capacity requirements is caused by using standard task times instead of considering time increments (case b), whereas

overestimations are due to fixing task interaction pairs as precedence relations (case a). We can thus conclude that completely ignoring task interactions bears a considerable risk that the resulting solution will be infeasible in practice. It seems to be generally more advisable, to model interaction pairs as precedence relationships, because less overestimations have occured. However, as can be seen, this approach holds a chance that capacity requirements are overestimated and resources are wasted.

6 Conclusions and future research

In this paper, we have introduced the concept of sequence-dependent time increments, a phenomenon which occurs in real-life assembly line balancing situations, but has not been considered in the literature thus far. Innovative mathematical formulations of this problem were discussed and presented.

Furthermore, two efficient search procedures were developed which make use of the extensive prior research in the field of ALB. Even problem instances of real-world size could be solved to optimality or at least near optimal solutions were found by using the effective solution procedures available for SALBP, like SALOME, while standard MIP solvers proved to be inefficient.

We believe that a lot of practice-relevant ALB extensions can be tackled by flexible search procedures based on the repeated solution of SALBP-instances. In the light of the high performance of existing solution procedures for SALBP, an identification of further ALB extensions with similar characteristics promises to be fruitful.

It is very likely that the solution quality of such approaches will easily surpass standard MIP solvers, while the effort of implementation is reduced in comparison to more specialized solution procedures.

Appendix

To examine the statistical significance of the performance differences of the solution procedures within the experiments documented in Section 5.2 and 5.3, we employ the non-parametrical Wilcoxon Signed Rank Test (WSRT). It is used instead of the more popular Paired Student's t Test, because the differences in relative deviations obtained by any two procedures A and B compared are not normally distributed as has been analysed prior to choosing the test method.

In a first step, the WSRT computes and ranks the absolute differences of two related samples (each sample consists of the relative deviations obtained by either method for all instances of the data set). A negative (positive) difference indicates that A produces a smaller (larger) deviation than B. In a second step, the sum of ranks assigned to the negative differences and the sum of ranks assigned to the positive differences are computed. Ties, i.e., instances where both methods obtain the same objective function value, are ignored. Table 11 displays all comparisons of two procedures A and B made (with the pairs derived from the overall ranking of the procedures based on the average

Table 11 Wilcoxon Signed Rank Test statistic

	A = basic, B = advanced		A = L1, B = basic		A = L2, B = L1	
	n	sum of ranks	n	sum of ranks	n	sum of ranks
A better (negative difference)	15	637.00	8	13,620.00	42	1,255.50
B better (positive difference)	91	5,034.00	1,698	1,442,451.00	36	1,825.50
A and B equal (zero difference)	4,198	–	2,598	–	4,226	–
$z\|p-value$	−6.935	0.000	−35.109	0.000	−1.423	0.155

relative deviations). The columns "n" specify the numbers of instances where A is better (negative difference), B is better (positive difference) and both are equal (tie). The other columns display the sums of the corresponding ranks.

Under the null-hypothesis H_0 that the median of the differences is zero, the distribution of the differences is expected to be symmetric around zero with randomly distributed positive and negative differences. Hence, the probability of observing a certain sum of positive or negative ranks (or a higher value) for a given sample size can be used to test H_0.

The last row lists the empirical z-values based on negative ranks and the two-tailed significance (p-value) which is for large sample sizes asymptotically estimated by a normal distribution. A p-value smaller than 0.05 suggests a rejection of the null-hypothesis. So, we can state that the advanced search procedure outperforms the basic one and both search procedures outperform the model L1 significantly. The overall performance of the two IP-models L1 and L2 is not found to be significantly different.

References

Baybars I (1986) A survey of exact algorithms for the simple assembly line balancing problem. Manage Sci 32:909–932

Becker C, Scholl A (2006) A survey on problems and methods in generalized assembly line balancing. Eur J Oper Res 168:694–715

Bockmayr A, Pisaruk N (2001) Solving assembly line balancing problems by combining IP and CP. In: Proceedings of the 6th Annual Workshop of the ERCIM Working Group on Constraints, Prague, Czech Republic

Bowman EH (1960) Assembly-line balancing by linear programming. Oper Res 8:385–389

Boysen N, Fliedner M, Scholl A (2006) A classification of assembly line balancing problems. Eur J Oper Res (in press)

Erel E, Sarin SC (1998) A survey of the assembly line balancing procedures. Prod Plan Control 9:414–434

Ghosh S, Gagnon RJ (1989) A comprehensive literature review and analysis of the design, balancing and scheduling of assembly systems. Int J Prod Res 27:637–670

Hackman ST, Magazine MJ, Wee TS (1989) Fast, effective algorithms for simple assembly line balancing problems. Oper Res 37:916–924

Hoffmann TR (1992) EUREKA: a hybrid system for assembly line balancing. Manage Sci 38:39–47

Johnson RV (1988) Optimally balancing large assembly lines with "FABLE". Manage Sci 34:240–253
Lübke M (2006) Methoden der Präferenzmessung: Formale Analyse, Vergleich und Weiterentwicklung. Diploma Thesis, University of Jena
Patterson JH, Albracht JJ (1975) Assembly-line balancing: Zero-one programming with Fibonacci search. Oper Res 23:166–172
Peeters M, Degraeve Z (2006) An linear programming based lower bound for the simple assembly line balancing problem. Eur J Oper Res 168:716–731
Pinnoi A, Wilhelm WE (1997) A family of hierarchical models for assembly system design. Int J Prod Res 35:253–280
Rekiek B, Dolgui A, Delchambre A, Bratcu A (2002) State of art of optimization methods for assembly line design. Annu Rev Control 26:163–174
de Reyck B, Herroelen W (1997) Assembly line balancing by resource-constrained project scheduling—A critical appraisal. Found Comput Control Eng 22:143–167
Saltzman MJ, Baybars I (1987) A two-process implicit enumeration algorithm for the simple assembly line balancing problem. Eur J Oper Res 32:118–129
Scholl A (1993) Data of assembly line balancing problems. Schriften zur Quantitativen Betriebswirtschaftslehre 16/93, TU Darmstadt
Scholl A (1999) Balancing and sequencing assembly lines, 2nd edn. Physica, Heidelberg
Scholl A, Becker C (2006) State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. Eur J Oper Res 168:666–693
Scholl A, Klein R (1997) SALOME: A bidirectional branch and bound procedure for assembly line balancing. Inf J Comput 9:319–334
Scholl A, Klein R (1999) Balancing assembly lines effectively—a computational comparison. Eur J Oper Res 114:50–58
Scholl A, Voß S (1996) Simple assembly line balancing—Heuristic approaches. J Heuristics 2:217–244
Sprecher A (1999) A competitive branch-and-bound algorithm for the simple assembly line balancing problem, Int J Prod Res 37:1787–1816
Talbot FB, Patterson JH, Gehrlein WV (1986) A comparative evaluation of heuristic line balancing techniques. Manage Sci 32:430–454
Thangavelu SR, Shetty CM (1971) Assembly line balancing by zero-one integer programming. AIIE Trans 3:61–68
Ugurdag HF, Rachamadugu R, Papachristou CA (1997) Designing paced assembly lines with fixed number of stations. Eur J Oper Res 102:488–501
Wee TS, Magazine MJ (1982) Assembly line balancing as generalized bin packing. Oper Res Lett 1:56–58
White WW (1961) Comments on a paper by Bowman. Oper Res 9:274–276
Williams HP (1999) Model building in mathematical programming, 4th edn. Wiley, New York