OR Spectrum (2013) 35:291–320
DOI 10.1007/s00291-011-0265-0

REGULAR ARTICLE

The assembly line balancing and scheduling problem with sequence-dependent setup times: problem extension, model formulation and efficient heuristics

Armin Scholl · Nils Boysen · Malte Fliedner

Published online: 1 July 2011

Abstract Assembly line balancing problems (ALBP) consist of distributing the total workload for manufacturing any unit of the products to be assembled among the work stations along a manufacturing line as used in the automotive or the electronics industries. Usually, theory assumes that, within each station, tasks can be executed in an arbitrary precedence-feasible sequence without changing station times. In practice, however, the task sequence may influence the station time considerably as sequence-dependent setups (e.g., walking distances, tool changes) have to be considered. Including this aspect leads to a joint balancing and scheduling problem, which we call SUALBSP (setup assembly line balancing and scheduling problem). In this paper, we modify the problem by modeling setups more realistically, give a new, more compact mathematical model formulation and develop effective heuristic solution procedures. Computational experiments based on existing and new data sets indicate that the new procedures outperform formerly proposed heuristics. They are able to solve problem instances of real-world size with small deviations from optimality in computation times short enough to be accepted in real-world decision support systems.

A. Scholl (✉)
Chair of Management Science, Friedrich-Schiller-University of Jena,
Carl-Zeiß-Straße 3, 07743 Jena, Germany
e-mail: armin.scholl@uni-jena.de

N. Boysen · M. Fliedner
Chair of Operations Management, Friedrich-Schiller-University of Jena,
Carl-Zeiß-Straße 3, 07743 Jena, Germany
e-mail: nils.boysen@uni-jena.de

M. Fliedner
e-mail: malte.fliedner@uni-jena.de

Keywords Assembly line balancing · Mass-production · Combinatorial optimization · Setup time · Scheduling

1 Introduction

Setup times are regularly considered as being sequence-dependent in job shop settings and other low volume production systems (cf. Allahverdi et al. 2008). The rationale behind this is that products are typically assumed to be so diverse, that additional configuration effort arises between any two jobs. In contrast to that, sequence-dependent setup times are hardly ever considered in the operational planning of paced assembly systems which are typical in the mass-production of standardized commodities. Even if different product models are assembled (in a mixed-model assembly), setup times between jobs are assumed to be negligible due to the use of flexible machinery and the high amount of standardized operations. But even though setup times do not depend on the sequence of jobs (models), there are in fact several practical settings in which setup times are incurred based on the sequence of tasks assigned to the operators of the line:

- In automotive assembly lines (and other assembly lines manufacturing large workpieces), work can be performed at different (mounting) positions inside and outside the car body (cf. Becker and Scholl 2009) such that walking distances are relevant and depend on the positions where two consecutive tasks are executed. Due to the size of a car body, those distances might be zero (same position) or amount up to about 10 m (if a worker must walk to the opposite side of the car body). As walking speed is typically assumed to be about 1 m/s (cf., e.g., Barnes 1959, p. 376; Kanawaty 1992, pp. 297), we might have walking-related setup times of 0 up to 10 s per setup. Concerning usual cycle times of about 60–90 s, this is a very critical aspect.
- Usually, walking distances are increased due to the necessity to fetch parts from material containers placed along the moving conveyor system. Furthermore, times to withdraw the parts from the container are to be considered. At a major German car manufacturer, (unproductive) walking and withdrawal times amount to about 10–15% of total production time. As a consequence, it is quite necessary to consider those times in a sequence-dependent manner in order to build station loads with small unproductive times.
- Regularly, tools are required to perform tasks. If consecutive tasks require different tools it will be necessary to change them causing a setup time or the same tool can be used anymore such that no setup is required. These sequence-dependent setups are typical in robotic assembly lines where robots can use a single tool at a time and have to change them if required (cf. Andrés et al. 2008). However, this is even necessary in manual assembly since most tasks require additional equipment like power screwdrivers, elevating mechanisms and fit-up aids.
- If two consecutive tasks require the workpiece in different positions (e.g., height, angle) it will be necessary to turn or lift the workpiece between performing those tasks causing a sequence-dependent setup time.

– If time is required for curing or cooling processes (e.g., after gluing or melting), then tasks to be executed at a corresponding mounting position must wait until this process is completed.

As a consequence, it is necessary to consider setup times even when balancing paced assembly line production systems. As practice regularly demands for fast and easy-to-implement heuristics, we do not focus on computationally expensive search procedures but try to develop a procedure which is able to solve even large problem instances quickly while achieving a high solution quality. Since assembly line balancing is a medium-term planning task, the available time for finding a good solution would seem to be less critical in principle. However, it has to be considered that real-world problem instances can be extremely large (up to several hundreds or even thousands of tasks) and that planners usually have to examine a considerable number of alternative line balancing scenarios to incorporate uncertain model mix information or different technical or logistical conditions. Furthermore, planners tend to expect that solution proposals are computed almost instantaneously, so that computation times are of crucial importance with regard to the acceptability of a planning tool. In recent projects with major car manufacturers, the aforementioned reasons led to the decision to implement priority rule-based balancing algorithms as part of a decision support tool instead of more elaborate procedures.

The paper on hand contributes to closing this gap and is organized as follows: In Sect. 2, we introduce the classical assembly line balancing problem and review the very restricted approaches to consider setup times in assembly line balancing proposed by former research papers. Section 3 defines a new joint balancing and scheduling problem (SUALBSP) which fully integrates setup times in assembly line balancing by differentiating between forward and backward setups relevant in practice. A mathematical model for this problem is formulated in Sect. 4. Heuristics formerly proposed for a restricted version of the problem are described and appropriately extended in Sect. 5. A new composite heuristic with a number of possible components is developed in Sect. 6 and tested in extensive computational experiments in Sect. 7. The final Sect. 8 summarizes the paper and analyzes future research needs.

2 Review of relevant research

An assembly line consists of *(work) stations* $k = 1, \ldots, m$ arranged along some type of conveyor belt. The workpieces are consecutively launched down the line and are moved from station to station. At each station, certain operations are repeatedly performed regarding the constant *cycle time* available per work-cycle. The decision problem of optimally partitioning the assembly work among the stations with respect to some objective is known as the *Assembly Line Balancing Problem* (*ALBP*); cf., e.g., Baybars (1986); Becker and Scholl (2006); Boysen et al. (2007).

The total amount of work is divided into a set of *tasks* $V = \{1, \ldots, n\}$ which build the nodes of a precedence graph. Performing a task i takes a *task time* (node weight) t_i; the total production time per product unit is $t_{\text{sum}} = \sum_{i \in V} t_i$. Technological and organizational conditions require to observe *precedence relations* (i, j) between different tasks i and j. Non-redundant precedence relations are represented as arcs

in the precedence graph. In order to simplify the presentation, we assume that graph $G = (V, E, t)$ is acyclic and numbered topologically.

An ALBP generally consists of finding a feasible *line balance*, i.e., an assignment of each task to a station such that the cycle time constraints, the precedence constraints and possible further restrictions are fulfilled. The set S_k of tasks assigned to a station k constitutes its *station load*. Usually, it is assumed that the *station time* (required working time in each cycle) which must not exceed the cycle time, simply results from summing up the execution times of all tasks in the load, i.e., $t(S_k) = \sum_{i \in S_k} t_i$. In case of $t(S_k) < c$, the station k has an *idle time* of $c - t(S_k)$ time units in each cycle.

The most popular ALBP is called *Simple Assembly Line Balancing Problem (SALBP)*. It considers a single product, a serial line, fixed task times and processes as well as equally equipped and manned stations (e.g., Baybars 1986; Scholl 1999; Boysen et al. 2007). The main problem parameters are the number m of stations and the cycle time c which can be given as a constant or considered as a variable defining several problem versions. The most relevant version referred to as SALBP-1 ([| |m] in the classification scheme of Boysen et al. 2007) is to minimize m given c. This problem and the other versions are NP-hard. Recent surveys covering SALBP models and procedures are given by Erel and Sarin (1998), Scholl (1999, ch. 2,4,5), Rekiek et al. (2002) as well as Scholl and Becker (2006).

Despite of setup times being very relevant in real-world assembly systems as stated in Sect. 1, assembly line balancing problems with sequence-dependent setup times have only been considered by a few research papers: Andrés et al. (2008) extend the classical SALBP-1 by additionally considering sequence-dependent setup times. They formulate a binary linear program and propose priority-rule-based and GRASP procedures. A large number of additional priority-rule-based procedures are developed and examined by Martino and Pastor (2010). Futhermore, see Arcus (1966), Wilhelm (1999) as well as Bautista and Pereira (2002).

Former research considers setup times only indirectly via assignment restrictions (e.g., Boysen et al. 2007; Scholl et al. 2010) along the following reasoning: If setup times are known to be (too) large, planners might take this into account by setting the related tasks incompatible, i.e., they must not be assigned to the same station (e.g., Becker and Scholl 2009). While this policy effectively ensures that setups are avoided it also comes at severe disadvantages. Strict incompatibilities reduce degrees of freedom unnecessarily and thereby lead to a systematic overestimation of required capacities, which becomes the more significant, the more assignment restrictions have to be considered. As a consequence, task incompatibilities should be reserved for unduly high setup times exclusively. Then, the (relevant) smaller setup times remain in the problem such that this approach is not useful whenever setup times are not negligible. In this work, we will therefore propose models and algorithms that directly take setup times into account. We will further evaluate our results against approximate approaches that include setups in a relaxed manner, in order to investigate the benefits of an exact planning in more detail.

The problem SDALBP examined by Scholl et al. (2008) also considers sequence-dependent time components. Nevertheless, the problems differ considerably with regard to their practical motivation and mathematical structure. Setup times are caused

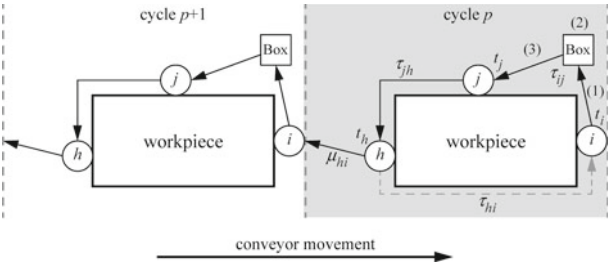


Fig. 1 Consecutive cycles and their connection at a single workstation

by direct succession of two tasks at the same workstation, whereas time increments of SDALBP are due to interactions among task effects (e.g. hindering of mounting a part by already installed components), irrespective of the workstation to which a preceding task is assigned.

3 The setup assembly line balancing and scheduling problem (SUALBSP)

We extend the approach of Andrés et al. (2008) by additionally distinguishing between forward and backward setups. The term *forward setup* refers to a situation where task j is executed directly after task i in the *same cycle*, i.e., at the same workpiece, observing a (forward) setup time $\tau_{ij} \geq 0$. A *backward setup* occurs if task i is the last one executed at the workpiece of a cycle p and the worker has to move to the *next workpiece* which is to be assembled in cycle $p+1$. This transfer causes a (backward) setup time $\mu_{ij} \geq 0$ which must be finished by the end of cycle p in order to start execution of task j just when cycle $p+1$ begins. Note that since stations are supposed to be independent and exclusively operated by a single (team of) worker(s), forward and backward setups are only considered among tasks at the same station, not between adjacent stations.

Figure 1 contains an example which shows that forward and backward setup times will generally be unequal in the real-world. We consider a station with ordered load $\langle i, j, h \rangle$, i.e., the tasks i, j, and h are performed in a cyclic manner on consecutive workpieces (e.g., car bodies). Each cycle p starts at time 0 with execution of task i which consumes time t_i. Afterwards, the operator has to walk to a box (1) where he has to fetch a part needed for task j (2). From this box, he has to move to the mounting position of task j (3). Summing up those time components results in setup time τ_{ij} to change over from task i to j which, thus, starts at time $t_i + \tau_{ij}$ and ends t_j time units later. Afterwards, the operator moves to the mounting position of task h (perhaps he fetches a further part or a tool on the way) which takes time τ_{jh}. After having executed h for t_h time units, he has to travel to the succeeding workpiece in cycle $p+1$. This takes backward setup time μ_{hi} and the current cycle p is finished. In order to be able to manage all productive and unproductive steps within each cycle, the following condition (*cycle time constraint*) has to be fulfilled: $t_i + \tau_{ij} + t_j + \tau_{jh} + t_h + \mu_{hi} \leq c$.

Assume that another ordered station load, say, $\langle j, h, i \rangle$ has to be considered. Then, a forward setup from h to i with time τ_{hi} (dashed arc), which is obviously different from μ_{hi} (cf. Fig. 1), takes place. By the way of contrast, the model of Andrés et al.

(2008) does not distinguish between the loads $\langle i, j, h \rangle$ and $\langle j, h, i \rangle$, because they assume that the same setup times are valid in both directions, i.e., $\mu_{ij} = \tau_{ij}$ for all task pairs $i, j \in V$. This includes $\mu_{ii} = \tau_{ii} = 0$ for all tasks i which is fairly inappropriate for our example, where τ_{ii} is not relevant and μ_{ii} reflects the transfer to the next workpiece. Obviously, this assumption of Andrés et al. (2008) considerably restricts applicability of their model in practice to cases where setups are not related to worker or workpiece movement.

For our model, we assume that the (modified) *triangle inequality* is valid, i.e., the station time is assumed to monotonically increase when an additional task is included. This condition is fulfilled if the following inequalities hold for all task triplets $i, j, h \in V$:

$$\tau_{ih} + t_h + \tau_{hj} \geq \tau_{ij} \quad \text{and} \quad \tau_{ih} + t_h + \mu_{hj} \geq \mu_{ij} \quad \forall i, j, h \in V \tag{1}$$

The triangle inequality will be fulfilled in almost any case in practice:

- If a tool (or the workpiece position) has to be changed between tasks i and j with setup time τ_{ij}, it might be possible that inserting task h allows for using the tool of task i, i.e., $\tau_{ih} = 0$. Then, however, the change must be performed between h and j, i.e., $\tau_{hj} = \tau_{ij}$, and the triangle inequality is valid due to $t_h \geq 0$.
- If setups are caused by walking distances, the triangle inequality obviously holds even if the intermediate task h might be on the way from i to j, i.e., $\tau_{ij} = \tau_{ih} + \tau_{hj}$.
- Usually, material boxes are positioned after the line balance and task schedules are known. So, they can be located such that walking distances are minimized.
- If cooling or curing takes place between tasks i and j, this waiting time can usually not be reduced by performing an intermediate task h (at another mounting position).

Finally, it is a necessary pre-condition for *feasibility* that each task, taken as the single element of a station load, can be executed within the cycle time, i.e., $t_i + \mu_{ii} \leq c$ $\forall i \in V$.

The new problem SUALBSP now can be defined as follows: Build a minimal number of ordered station loads such that each task is assigned to exactly one station, the precedence constraints are observed and the cycle time is not exceeded in any station when considering cyclic task execution with sequence-dependent (forward and backward) setup times. In the classification scheme of Boysen et al. (2007), SUALBSP is represented by the tuple $[\Delta t_{dir} | \quad | m]$.

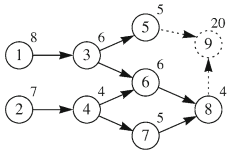
4 Model for SUALBSP

We formulate the SUALBSP as a mixed-binary linear model which combines the logics of the traditional model for SALBP Scholl (1999) and the formulation of Miller et al. (1960) for the traveling salesman problem. Table 1 summarizes already introduced notation and defines additional parameters and variables.

Example The problem and some of the notations defined in Table 1 are clarified by means of an example with the cycle time set to $c = 20$ and the precedence graph as

Table 1 Notation

V	Set of tasks; $V = \{1, \ldots, n\}$
E ($E*$)	Set of direct (all) precedence relations
t_i	Execution time for task $i \in V$ with $t_{\text{sum}} = \sum_{i \in V} t_i$
τ_{ij} (μ_{ij})	Forward (backward) setup time from task i to j
P_i (P_i^*)	Set of direct (all) predecessors of task $i \in V$
F_i (F_i^*)	Set of direct (all) successors/followers of task $i \in V$
$\overline{m}$	Upper bound on the number of stations, e.g., $\overline{m} = n$ or heuristic solution value
E_i ($L_i(\overline{m})$)	Earliest (latest station for task $i \in V$; $E_i := \left\lceil \frac{t_i + \sum_{j \in P_i^*} t_j}{c} \right\rceil$; $L_i(\overline{m}) := \overline{m} + 1 - \left\lceil \frac{t_i + \sum_{j \in F_i^*} t_j}{c} \right\rceil$)
K	Set of possible stations; $K = \{1, \ldots, \overline{m}\}$
FS_i	Set of stations to which task $i \in V$ is feasibly assignable; $FS_i = \{E_i, E_{i+1}, \ldots, L_i(\overline{m})\}$
B_k	Set of tasks assignable to station $k \in K$; $B_k = \{i \in V \mid k \in FS_i\}$
F_i^τ (P_i^τ)	Set of tasks which may directly follow (precede) task i in a station load in forward direction; $F_i^\tau = \{j \in V - (F_i^* - F_i) - P_i^* - \{i\} \mid FS_i \cap FS_j \neq \emptyset\}$; $P_i^\tau = \{h \in V \mid i \in F_h^\tau\}$
F_i^μ (P_i^μ)	Set of tasks which may directly follow (precede) task i in a station load in backward direction; $F_i^\mu = \{j \in V - F_i^* \mid FS_i \cap FS_j \neq \emptyset\}$; $P_i^\mu = \{h \in V \mid i \in F_h^\mu\}$
M (ε)	Sufficiently large (small) number, i.e., $M = n \cdot c$ and $\varepsilon = 1/(c \cdot (n+1))$
x_{ik}	Binary variable with value 1, iff task $i \in V$ is assigned to station $k \in FS_i$
y_{ij}	Binary variable with value 1, iff task $i \in V$ is direct predecessor of $j \in F_i^\tau$ in a station load
w_{ij}	Binary variable with value 1, iff task $i \in V$ is last and $j \in F_i^\mu$ is first task in a station load
z_i	Continuous variable for encoding the number of the station task $i \in V$ is assigned to
f_i	Continuous variable for start time of task $i \in V$ (relative to launch time at the *first* station)
T_k	Continuous variable for station time of station $k \in K$



τ	1	2	3	4	5	6	7	8
1	-	1	2	1	-	-	-	-
2	1	-	3	2	1	-	-	-
3	-	3	-	1	2	3	1	-
4	1	-	1	-	1	2	0	-
5	-	1	-	1	-	1	1	2
6	-	-	-	-	1	-	2	3
7	-	-	-	-	1	2	-	1
8	-	-	-	-	2	-	-	-

μ	1	2	3	4	5	6	7	8
1	4	5	-	3	-	-	-	-
2	3	4	1	-	3	-	-	-
3	6	7	4	5	-	-	5	-
4	5	6	3	4	5	-	-	-
5	4	5	2	3	4	5	3	2
6	3	4	1	2	3	4	2	-
7	5	6	3	4	5	6	4	-
8	6	7	4	5	6	7	5	4

Fig. 2 Data of the example problem (cycle time $c = 20$)

well as forward and backward setup times given in Fig. 2. The node weights denote the task times with $t_{\text{sum}} = 45$. Node 9 is a dummy sink node—required for technical reasons in the model as explained below—with its task time t_9 set to the cycle time and all its setup times set to zero (they are thus omitted in the matrices).

The setup time matrices have been defined such that the triangle inequality is fulfilled using the method explained in Sect. 7.1 so as to reflect that, in practice, setup times are often due to walking distances. The omitted values in matrix τ indicate that a forward setup can only represent a changeover from a task i to another task j that is not related to i by precedence or is a direct successor. These tasks are collected in

set F_i^{τ}. Backward setups from i are restricted to tasks $j \in F_i^{\mu}$ which are not successors of i. As defined in Table 1, these sets could be further restricted by means of feasible station sets FS_i whenever two tasks cannot be assigned to the same station (in a solution with no more than $\overline{m}$ stations). Note that these irrelevant setup times can be set to $c + 1$ or another sufficiently large value. Though not relevant for the model, this data manipulation is helpful in programming solution procedures as it renders an explicit examination of the above mentioned sets superfluous.

Using the defined notation, we get the model (2)–(16). The objective function (2) minimizes the index of the station to which the unique sink node n is assigned and, thus, the total number of stations required.[1] As a secondary objective the values of the variables T_k are minimized in order to get station times with minimum setups. Due to setting ε sufficiently small, the secondary objective does not influence minimization of the number of stations.

The assignment of each task i to exactly one station $k \in FS_i$ is guaranteed by (3) and transformed into the corresponding station index z_i by (4). The constraint sets (5) and (6) ensure that each task i is followed and preceded by exactly one other task j in the cyclic sequence of a station load. In combination with (7) these constraints demand that in each cycle exactly one of the relations is a backward setup. Tasks contained in the same cycle are forced to be assigned to the same station by constraints (8) and (9) which are to be fulfilled pairwise as equations ($z_i = z_j$), if $y_{ij} = 1$ and $w_{ij} = 1$, respectively, and are redundant, otherwise. The precedence relations are fulfilled through (10), while the forward ordering of tasks in a station load is guaranteed by (11). Each task i must be executed completely in the time interval $[(z_i - 1) \cdot c, z_i \cdot c]$ assigned to its station z_i (time elapsed since workpiece was launched down the line) by the restrictions (12) and (13). In (13) it is sufficient to ensure that the final backward setup (indicated by $w_{ij} = 1$) will be finished in-time. The station time T_k is computed by (14) which is—just as (13)—only binding for the task i that is the last one in station k.

$$\text{Minimize} \quad m(\mathbf{x}, \mathbf{y}, \mathbf{z}) = z_n + \varepsilon \cdot \sum_{k \in K} T_k \qquad \text{subject to} \tag{2}$$

$$\sum_{k \in FS_i} x_{ik} = 1 \qquad \forall i \in V \tag{3}$$

$$z_i = \sum_{k \in FS_i} k \cdot x_{ik} \qquad \forall i \in V \tag{4}$$

$$\sum_{j \in F_i^{\tau}} y_{ij} + \sum_{j \in F_i^{\mu}} w_{ij} = 1 \qquad \forall i \in V \tag{5}$$

$$\sum_{i \in P_j^{\tau}} y_{ij} + \sum_{i \in P_j^{\mu}} w_{ij} = 1 \qquad \forall j \in V \tag{6}$$

[1] If no unique terminal node exists, a dummy sink node has to be introduced and defined as the successor of the original terminal nodes. In order to compute setup times in the last station in a correct manner, its task time is set to c and all its setup times to zero. This ensures that only the dummy node is assigned to the (dummy) last station z_n. In order to remove this unnecessary station and to get a correct objective value in this case, the value of z_n has to be reduced by 1.

$$\sum_{i \in V} \sum_{j \in F_i^{\mu}} w_{ij} = z_n \tag{7}$$

$$z_i + M \cdot (1 - y_{ij}) \geq z_j \quad \text{and} \quad z_j + M \cdot (1 - y_{ij}) \geq z_i \qquad \forall i \in V,\, j \in F_i^{\tau} \tag{8}$$

$$z_i + M \cdot (1 - w_{ij}) \geq z_j \quad \text{and} \quad z_j + M \cdot (1 - w_{ij}) \geq z_i \qquad \forall i \in V,\, j \in F_i^{\mu} \tag{9}$$

$$f_j \geq f_i + t_i \qquad \forall (i, j) \in E \tag{10}$$

$$f_j \geq f_i + t_i + \tau_{ij} + M \cdot (y_{ij} - 1) \qquad \forall i \in V,\, j \in F_i^{\tau} \tag{11}$$

$$f_i \geq c \cdot (z_i - 1) \qquad \forall i \in V \tag{12}$$

$$f_i + t_i + \mu_{ij} \leq c \cdot z_i + M \cdot (1 - w_{ij}) \qquad \forall i \in V,\, j \in F_i^{\mu} \tag{13}$$

$$T_k + c \cdot (z_i - 1) \geq f_i + t_i + \sum_{j \in F_i^{\mu}} \mu_{ij} \cdot w_{ij} - M \cdot (1 - x_{ik}) \qquad \forall i \in V, k \in FS_i \tag{14}$$

$$x_{ik} \in \{0, 1\} \quad \forall i \in V, k \in FS_i; \quad y_{ij} \in \{0, 1\} \quad \forall i \in V,\, j \in F_i^{\tau};$$
$$w_{ij} \in \{0, 1\} \quad \forall i \in V,\, j \in F_i^{\mu} \tag{15}$$

$$f_i, z_i, T_k \geq 0 \quad \forall i \in V, k \in K \tag{16}$$

The variables are defined by (15) and (16). Among those variables, only x_{ik}, y_{ij}, w_{ij}, and f_i are required to define a complete and correct model. The other ones are introduced only in order to enhance readability and comprehensibility. The variables z_i and their definitions (4) can be removed from the model by replacing them in all constraints via (4). The variables T_k and the corresponding constraints (14) are only introduced to directly compute the station time which could also be retrieved from the other variables after having solved the model. In the objective function, the sum of station times could be replaced by the sum of starting times f_i of all tasks i to ensure continuous operation within stations.

In the most compact version, the model contains up to $2 \cdot n^2 + n \cdot \overline{m}$ binary variables and n continuous variables. Due to $\overline{m} \leq n$, the number of (binary) variables is bounded by $O(n^2)$. Furthermore, the model contains up to $7 \cdot n^2 + n \cdot \overline{m} + 5 \cdot n + 1$ constraints, i.e., the number of constraints is bounded by $O(n^2)$. The model of Andrés et al. (2008) contains $2 \cdot n^2 \cdot \overline{m} + n^2 + \overline{m}$, i.e., $O(n^3)$ binary variables and $2 \cdot n^3 \cdot \overline{m} + n^2 \cdot (\overline{m} + 1) + 2 \cdot (n + \overline{m}) + n$, i.e, $O(n^4)$ constraints. Provided that $n = 100$ tasks are to be assigned to at most $\overline{m} = 20$ stations, which would be a medium-sized problem instance in the real world, the new model contains up to 22,000 binary variables and 100 continuous ones as well as 72,501 constraints. The model of Andrés et al. (2008) would require up to 410,020 binary variables and 40,214,100 constraints. Though, in both models, (further) reductions (via FS_i and B_k) are possible, it is obvious that the new model's size is smaller at least by an order of magnitude.

5 Former heuristic procedures for SUALBSP

We shortly review and describe the few procedures for SUALBSP proposed in literature and their reported performance as the underlying ideas and experimental results inspired our development of a new composite heuristic.

5.1 Priority rule-based procedures (PRBP)

The application of PRBP to different ALBPs has a long tradition (cf., e.g., Talbot et al. 1986; Boctor 1995; Scholl and Voß 1996). A large variety of priority rules and scheduling schemes have been tested for SALBP and other problem versions. As a basic result, it could be established for SALBP and SUALBSP as well that the station-oriented scheduling scheme is to be preferred to the task-oriented one (cf. Scholl and Voß 1996; Martino and Pastor 2010). Both scheduling schemes order tasks by priority rules and successively assign *available* tasks, i.e., tasks whose predecessors have already been assigned. While the station-oriented scheme builds station loads strictly one after the other, the task-oriented one always opens a new station if the highest-priority task does not fit in a station already open.

For solving SUALBSP, the *station-oriented scheduling scheme* works as follows: Stations are considered in forward direction, beginning with station 1. For each station k considered, a load S_k (with n_k tasks ordered in the sequence π^k) is built by successively adding available tasks such that the station time $t(S_k) = T(\pi^k)$ computed by (17) does not exceed the cycle time. Among several candidate tasks, the one with highest priority is chosen. If no task can be added in this manner, the load is maximal and the next station is opened and filled in the same way, until a feasible assignment of all tasks is achieved. The number of station loads formed determines the value of the heuristic solution (upper bound *UB*).

$$T(\pi^k) = \sum_{p=1}^{n_k-1} (t_{\pi_p^k} + \tau_{\pi_p^k,\pi_{p+1}^k}) + t_{\pi_{n_k}^k} + \mu_{\pi_{n_k}^k,\pi_1^k} \tag{17}$$

Andrés et al. (2008) and Martino and Pastor (2010) tested a large set of possible priority rules. It was found out that criteria based on a single parameter (like task time) are less effective than composite rules combining several parameters. The best composite rule proposed by Martino and Pastor (2010) is based on priority values computed as follows:

$$MP_i(k, p) := 5.0 \cdot t_i + 0.3 \cdot \sum_{j \in F_i^*} (t_j + 3.9 \cdot \overline{\tau}_j) - 45.3 \cdot \tau_{\pi_{p-1}^k,i} \tag{18}$$

With $\tau_{\pi_{p-1}^k,i}$ we denote the setup time necessary to change over from the currently last task at the position $p-1$ of the already constructed partial sequence to the task i which is a candidate for the current position p. For the first position, we initialize with $\tau_{0,i} := 0$. The value $\overline{\tau}_j$ represents the average setup time of a successor task $j \in F_i^*$ such that $\sum_{j \in F_i^*}(t_j + 3.9 \cdot \overline{\tau}_j)$ represents an adaptation of the positional weight originally defined by (Helgeson and Birnie 1961). This average value is updated in each step as it only should cover possible setup operations between still unassigned tasks.

Martino and Pastor (2010) fine-tuned the weights in (18) by adapting them to a training data set (10% of the instances of the data set) with the nonlinear optimization method of Nelder and Mead (1965).

Among several modifications of the basic station-oriented scheme, a *re-optimization* of the partial sequence π^k right after each task assignment is particularly effective. This is done by local search with moves shifting tasks to other positions of the sequence. In each iteration, the station time minimizing shift is taken until a local optimum is reached.

With these two modifications of the station-oriented PRBP, Martino and Pastor (2010) achieved astonishingly good results and could clearly outperform the more elaborate procedure of Andrés et al. (2008) described in the next section. So, one aim of our experiments will be to examine whether the fine-tuning to a particular data set has led to a composite rule which is useful for other data sets, too, or represents a "self-fulfilling prophecy" only.

5.2 Greedy randomized adaptive search procedure (Task-GRASP)

As a single-pass PRBP might lead to weak solutions, it is typically recommended to perform multiple passes (e.g., Boctor 1995). In order not to depend on having available a sufficiently large set of different rules and to compensate structural deficits of existing rules, it is, moreover, useful to incorporate some level of randomness into the process.

A well-known meta-strategy with these properties is called *greedy randomized adaptive search procedure* (GRASP). In each pass, a randomized greedy algorithm, e.g., a PRBP with randomized priority rule, is applied to construct a feasible solution which is, then, improved by a local search procedure (e.g., Resende and Ribeiro 2003).

Andrés et al. (2008) apply GRASP to SUALBSP as follows (called *Task-GRASP* due to the random task selection): In all (of 5 or 10) passes the same priority function is utilized to define task priorities. In each pass, a solution is constructed following the station-oriented scheme. However, the task chosen to be assigned is not necessarily the one with highest priority but one with sufficiently high priority. This is achieved by setting a *priority threshold* which defines a subset of assignable tasks, the *restricted candidate set*, to be selected from randomly with equal probabilities. Having normalized the interval of priorities computed for all candidate tasks to length 1 (with value 0 for the best and value 1 for the worst task), the priority threshold is set to 0.3, i.e., only tasks whose priorities are among the best 30% of the entire priority interval are candidates for random assignment. As priority function it is proposed:

$$AMP_i(k,\, p) := t_i + \tau_{\pi^k_{p-1},i} \tag{19}$$

To each solution constructed in this manner, a *local search heuristic* is applied in order to find an improved solution. Considering the assigned tasks as a sequence with n positions, each precedence-feasible exchange of tasks between two positions is examined (by forming maximal station loads following this sequence). Among all those exchange moves, the best one is selected. This process is repeated until no improved solution (with less stations) is found by such a move.

5.3 Avalanche

The procedure Avalanche developed by Boysen and Fliedner (2008) is a flexible heuristic procedure for a large variety of ALBPs and can be modified to solve SUALBSP easily. It is an iterative *two-phase algorithm* based on task sequences. In the first phase of each iteration, a number of precedence-feasible sequences is constructed by an *ant colony approach* (for this meta-heuristic cf., e.g., Dorigo and Blum 2005) based on a PRBP with the task time as priority value.

In the second phase, every solution constructed in the first phase is optimally partitioned by a graph approach explained in detail and extended in Sect. 6.4. For the next iteration, the solutions obtained before are analyzed with respect to solution quality. This information is connected with the performed assignments of tasks to sequence positions and utilized to update the so-called pheromon matrix by standard formulae of ant colony approaches (cf., e.g., Dorigo and Stützle 2003) which results in intensified new trails and evaporation of older ones. When constructing the next sequence, each ant utilizes the pheromon values which are combined with the original priority values to form a composite measure for task selection. For details we refer to Boysen and Fliedner (2008).

6 A new composite heuristic procedure for SUALBSP

We propose a new composite heuristic for SUALBSP which makes use of some former contributions and new ideas as well. It is intended to be a good compromise between solution quality and computation times as required in a real-world planning environment as discussed in Sect.1. In the following, we describe the ingredients of our new approach.

6.1 Rule-GRASP

Analyzing the procedure of Andrés et al. (2008) shows that the selection approach is rather arbitrary as only a single priority rule is taken and the threshold seems to be rather wide. Furthermore, the priority function (19) seems not to be very promising as it interprets large setup times $\tau_{\pi^k_{p-1},i}$ as being favorable. In fact, these setup times should be minimized and, thus, subtracted as realized in the composite priority rule of Martino and Pastor (2010) defined in (18).

As the latter rule considers setup times and other time criteria in a reasonable manner, this fine-tuned rule seems indeed to be a favorable candidate but should be accompanied by other rules that incorporate the same criteria without weighting them in such a definite manner as well as other criteria to define a robust and effective heuristic. In order to achieve these goals, we define a modified GRASP approach, called Rule-GRASP, because it is not the tasks that are selected at random but the priority rules applied. Using the chosen rule, the *highest-priority* task is selected and assigned instead of any one with acceptable priority.

As priority rules we utilize the following ones which have shown some potential in solving SALBP-1 (cf.,e.g., Talbot et al. 1986; Scholl and Voß 1996) or SUALBSP (cf. Martino and Pastor 2010). These rules cover different aspects of the problem structure and appeared to be useful as a combination in preliminary experiments though they are not necessarily among the best single rules in the tests of Martino and Pastor (2010):

MaxTMinSU	Select a task with largest task time reduced by increase of setup times: $MT_i(k, p) := t_i - (\tau_{\pi^k_{p-1},i} + \mu_{i,\pi^k_1} - \mu_{\pi^k_{p-1},\pi^k_1})$ for each candidate task i
MaxTMinSUSlack	Select a task with largest reduced time divided by the number of available stations (slack): $MTS_i(k, p) := MT_i(k, p)/(L_i(\overline{m}) - E_i + 1)$ for each candidate task i
MaxF	Select a task with maximal number of direct followers: $MF_i := \lvert F_i \rvert$ for each candidate task i
MinSlack	Select a task with minimal slack or, equivalently, maximal value: $MS_i(k) := k - L_j(\overline{m})$
Random	Select one of the available tasks by chance (with uniform probability).
MPRule	Select a task with maximal value of $MP_i(k, p)$ computed by (18).

For SALBP, it has been shown effective to apply solution procedures to the *reverse precedence graph* also, e.g., Scholl 1999, ch. 7.2. This is also possible for SUALBSP provided that all problem data are reversed correctly, i.e., arcs of the precedence graph as well as forward and backward setup times. Finally, the solution to the reverse problem must be inverted to get a solution for the original problem. So, we might apply our Rule-GRASP in *forward* direction (to the original problem) and in *backward* direction (to the reverse problem) without algorithmic changes.

6.2 Re-optimization

By assigning tasks to a station load one after the other, a sequence for task execution is defined. However, this sequence need not be optimal as it has been built in a greedy manner. In opposite to SALBP, the station times depend on the actual sequence and it might be possible to reduce setup times by rearranging the tasks.

For this reason, Martino and Pastor (2010) apply a local search procedure to re-optimize the sequence always *after* having assigned a task (see Sect. 5.1). However, this re-optimization might come too late as demonstrated by the example of Fig. 2. Imagine that task 2 has already been assigned to the first station. When task 1 is inspected to be assigned to the same station load it has to be checked if the station time is not greater than $c = 20$. However, by applying (17) we get $T(\langle 2, 1]) = t_2 + \tau_{21} + t_1 + \mu_{12} = 7 + 1 + 8 + 5 = 21$ which is too large. So, task 1 will not be

assigned and task 4 is taken instead. Without overstraining this small example, this might prevent from finding the optimal solution.

As a consequence, we propose a different approach which first computes priorities for all tasks that are *available* to the current position p of station sequence π^k. Starting with the highest-priority task i, it is examined if i fits into the station. This is done in two steps:

1. If the station time, computed by (17) for the sequence with i simply assigned to the currently last position p, is not greater than the cycle time, task i is assigned to this last position without further effort. Note that only the change in setup times$\tau_{\pi^k_{p-1},i} + \mu_{i,\pi^k_1} - \mu_{\pi^k_{p-1},\pi^k_1}$ has to be computed to update the station time.
2. Only if this simple test fails, an enumerative procedure is started which systematically constructs all feasible sequences of the trial station load including task i. In order to find promising sequences first, the procedure starts with assigning the tasks in a "nearest-neighbor" manner, i.e., the next task is always chosen such that the setup time increase is minimal. Following this ordering, the precedence-feasible sequences are constructed in a lexicographic manner recursively. Whenever the station time exceeds the cycle time, a backtracking step is performed. Once a feasible sequence with its station time not exceeding the cycle time is found, the search can be stopped.

If a feasible sequence is found in step (1) or (2), the examined task i is assigned. Otherwise, the next task in descending priority ordering is tested in the same manner. This process is repeated until a task can be assigned or no available task remains. In the first case, the next position $p + 1$ of sequence π^k is examined; in the latter case, a new station $k + 1$ with i at the first position of sequence π^{k+1} is opened. In our example, this procedure would find out in step (2) that the sequence $\langle 1, 2]$ is feasible with $T(\langle 1, 2]) = t_1 + \tau_{12} + t_2 + \mu_{21} = 8 + 1 + 7 + 3 = 19$ and would assign task 1.

Our experiments indicate that the re-optimization is useful for sequences with up to seven to ten tasks, because this sequencing problem is a variant of the traveling salesman problem and, hence, NP-hard by itself. So, in order to define a fast overall heuristic, we apply this re-optimization only if necessary (in opposite to Martino and Pastor (2010) who propose a local search after each task assignment) and promising (for up to seven tasks in our tests). As the number of tasks per worker is seldom greater than 10 to 12 in practice, this is not a very hard limitation. Additionally, we restrict the computation time as preliminary experiments indicate that the trade-off between computational effort and (positive) effect of re-optimization is best if a very short time is spent (0.01 s per sequence to be re-optimized in our tests). In many cases, this time is sufficient to find a feasible sequence, whereas in others it might consume minutes (or even hours) to find one or to prove that none exists. This is due to the complexity of this sequencing subproblem of SUALBSP.

We decided against applying local search procedures such as developed by Martino and Pastor (2010) and Andrés et al. (2008) because preliminary experiments strongly indicate that those procedures have only limited effect as they typically get stuck soon in a local optimum and, on average, do not run faster than our enumerative procedure.

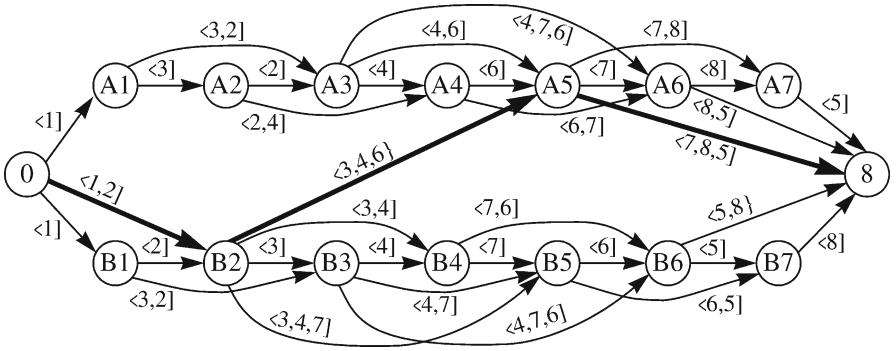


Fig. 3 Grouping graph

6.3 Fathoming

In a multi-pass procedure, time might be wasted by unnecessarily completing inferior partial solutions. In order to avoid this waste, we compute the minimal *productive* time (only execution times excluding setup times and idle times) that is necessary for the current station load in order to have still the potential to improve on the current best solution (incumbent solution). Let UB be the corresponding upper bound (number of stations in the incumbent solution) and $\mathrm{BD} := (\mathrm{UB} - 1) \cdot c - t_{\mathrm{sum}}$ the balance delay (total unproductive time) allowed in a solution with at most UB − 1 stations; then the lower bound on required productive time of the first station $k = 1$ is given by $\mathrm{MPT}_1 := \max\{0, c - \mathrm{BD}\}$. This time bound is successively increased for the next stations whenever unproductive times (setup times or idle times) are introduced, i.e., $\mathrm{MPT}_k := \mathrm{MPT}_{k-1} + c - \sum_{i \in S_{k-1}} t_i$ for $k > 1$.

The construction process is always immediately stopped whenever it becomes obvious that no load whose productive time is greater or equal than MPT_k can be achieved anymore. This fathoming enables to perform more passes in the same time such that improved solutions might be found faster.

6.4 Improvement by shortest-path computations

After having constructed a number of feasible solutions, these solutions might be combined to build improved solutions by extending the graph approach of Avalanche (cf. Sect. 5.3 as well as Boysen and Fliedner 2008).

This approach is best explained by means of our example of Fig. 2. Assume that two different solutions with four stations each have been obtained by applying Rule-GRASP, $A = \langle 1|3, 2|4, 6|7, 8, 5]$ and $B = \langle 1, 2|3, 4, 7|6, 5|8]$, with vertical bars indicating that a new station needs to be opened.

In order to re-optimize each sequence or to combine favorable parts of different (re-optimized) sequences, a so-called *grouping graph* is constructed (e.g.,Klein 1963; Easton et al. 1989; Boysen and Fliedner 2008; Boysen and Scholl 2009). Figure 3 shows the graph for our example which contains a node for each position p of each

sequence. This node represents the set of tasks assigned up to position p, e.g., node $A3$ represents feasible task subset $\{1, 2, 3\}$ and node 8 entire task set V. Additionally, a start node 0 which represents the empty set is introduced. These nodes are connected by arcs whenever their difference set forms a feasible station load. Due to the fact that each arc corresponds to one station, arc weights are all set to 1 and the minimal number of stations for each of the task sequences can be determined by finding the shortest path from node 0 to 8. In the example, additional options to combine tasks to station loads are found for each sequence by their grouping subgraphs (chains) which lead to additional solutions (e.g., $A' = \langle 1|3, 2|4, 6, 7|8, 5]$) but no overall improvement is achieved.

An improved solution with three stations can be determined by adding a *chain-connecting* arc as shown in Fig. 3: The nodes $B2 = \{1, 2\}$ and $A5 = \{1, 2, 3, 4, 6\}$ differ by the set $\{3, 4, 6\}$ which is a feasible station load (with sequence $\langle 3, 4, 6]$ and station time 20). With this arc on hand, a shortest path with three arcs is found, i.e., $\langle 1, 2|3, 4, 6|7, 8, 5]$ which is highlighted by bold arcs in the figure.

From a computational point of view, there are several means for saving time. Of course, nodes that represent the same task set can be merged, e.g., nodes $A4$ and $B4$ (which we have distinguished for presentation purposes only). As a consequence, all adjacent arcs are connected to this merged node. Having a look at the resulting graph shows that it is a subgraph of the state graph of a station-oriented dynamic programming (DP) approach (e.g., Jackson 1956; Easton et al. 1989). As a consequence, shortest paths can be computed just when nodes are constructed in the manner of a DP procedure. For example, right after generating merged node $A4 = B4$ it becomes obvious that its task set $\{1, 2, 3, 4\}$ can be assigned to two stations as the shortest path represents the partial solution $\langle 1, 2|3, 4]$. Furthermore, it is not necessary to search for arcs if a node will not be part of the shortest path. In our example, node B1 represents the non-maximal load with task 1 only. So, there is no need to identify the arc from B1 to B3.

The latter aspect is more relevant than it seems to be as the detection of arcs is not as trivial as in case of SALBP, where it is sufficient to sum up task times, but requires to decide whether there exists a feasible sequence of the tasks contained in the difference set. This NP-hard subproblem should not be solved unnecessarily. In our approach, we use the same two-step procedure to examine loads as described in Sect. 6.2. An arc is only introduced if it was possible to find a feasible sequence within the given limitations.

6.5 Lower bounds and data reduction

Since optimal solutions are not known for any problem instance tested and to evaluate solution capabilities of the heuristic solution procedures examined more deeply, we use an extended version of the lower bound argument defined by Andrés et al. (2008). This bound argument is an extension of the well-known *capacity bound* for SALBP-1 defined as follows:

$$\mathrm{LB1} := \left\lceil \frac{t_{\mathrm{sum}}}{c} \right\rceil \quad \text{with} \quad t_{\mathrm{sum}} = \sum_{j \in V} t_j \tag{20}$$

LB1 has been adapted to their restricted version of SUALBSP (setup times τ_{ij} used in both directions and $\tau_{ii} = 0\ \forall i \in V$) by Andrés et al. (2008) and corrected in Pastor et al. (2010). They compute the bound value, which we call $\mathrm{LB1}^f_{\mathrm{AMP}}$, in an iterative manner based on $\tau^q_{\mathrm{sum}}(\mathrm{RV})$, the sum of the q smallest setup times τ_{ij} with $i \in V$ and $j \in \mathrm{RV} - \{i\}$.

In order to explain the logic of the bound within the usual categories, we present it as a destructive improvement bound (cf. Klein and Scholl 1999; Scholl and Becker 2006) as follows: The bound is based on the fact that at least $q = n - (m-1) = n + 1 - m$ tasks must share a station with other tasks if (at most) m stations are available.[2] Thus, for (at most) m stations τ^q_{sum} is a lower bound on the total setup time required. If the total time capacity $TC(m) = m \cdot c$ provided by m stations is not sufficient to perform all work (t_{sum}) and the setups at least required (τ^q_{sum}), then m stations are not sufficient (this trial bound value is "destroyed"), and $m := m + 1$ is considered as trial number of stations in the next iteration. This process starts with $m = \mathrm{LB1}$ and is repeated until the current m cannot be destroyed. The final bound value is set to $\mathrm{LB1}^f_{\mathrm{AMP}} := m$. Unlike the procedure described in Pastor et al. (2010), our destructive improvement procedure increases the bound value in each iteration as it leaves out superfluous computations. It can be summarized in the following formula:

$$\mathrm{LB1}^f_{\mathrm{AMP}} := \min\{m \geq LB1 \mid t_{\mathrm{sum}} + \tau^{n+1-m}_{\mathrm{sum}} \leq \mathrm{TC}(m)\} \tag{21}$$

For SUALBSP, the lower bound argument $\mathrm{LB1}^f_{\mathrm{AMP}}$ can be adapted and improved by distinguishing between forward and backward setups as well as observing that not all tasks can be adjacent to each other in feasible sequences (cycles). Given a trial number m of stations, it is known that a backward setup is necessary in each station and at least $q = n - m$ forward setups are required. Let τ^q_{sum} now denote the sum of the q smallest forward setup times τ_{ij} with $i \in V$ and $j \in F^{\tau}_i$. Furthermore, let μ^q_{sum} be defined as the sum of the q smallest backward setup times μ_{ij} with $i \in V$ and $j \in F^{\mu}_i$; then the modified bound is computed as follows:

$$\mathrm{LB1}^{\mathrm{fb}}_{\mathrm{AMP}} := \min\{m \geq \mathrm{LB1} \mid t_{\mathrm{sum}} + \tau^{n-m}_{\mathrm{sum}} + \mu^{m}_{\mathrm{sum}} \leq \mathrm{TC}(m)\} \tag{22}$$

Example In our problem instance of Fig. 2 we have $t_{\mathrm{sum}} = 45$ and, thus, $\mathrm{LB1} = \lceil 45/20 \rceil = 3$. If only the matrix τ is taken for both directions without eliminating irrelevant entries (as already done in Fig. 2), the originally proposed bound $\mathrm{LB1}^f_{\mathrm{AMP}}$ will have no effect for $m = 3$ due to $\tau^{n+1-m}_{\mathrm{sum}} = 0$ as eight original τ_{ij} entries are zero. Though no bound improvement can be achieved for this example (as the available total unproductive time is $\mathrm{BD}(m = 3) = 3 \cdot 20 - 45 = 15$), the modified bound takes advantage of distinguishing between forward and backward setups and removing irrelevant values, i.e., $\tau^5_{\mathrm{sum}} = 0 + 4 \cdot 1 = 4$ and $\mu^3_{\mathrm{sum}} = 1 + 1 + 2 = 4$.

The logic behind the AMP-bound can be applied more directly to identify sequence-independent parts of the setup times by (23) and (24) in order to increase task times, if possible.

[2] This lower bound on setup operations between different tasks is achieved if each of $m-1$ stations contains a single task and the remaining station contains the residual q tasks.

$$\omega_i := \min\{\min\{\tau_{ij} \mid j \in F_i^\tau\},\ \min\{\mu_{ij} \mid j \in F_i^\mu\}\} \tag{23}$$

$$\alpha_i := \min\{\min\{\tau_{hi} - \omega_i \mid h \in P_i^\tau\},\ \min\{\mu_{hi} - \omega_i \mid h \in P_i^\mu\}\} \tag{24}$$

After having executed task i, at least ω_i time units must be spent before another task can be started. As a consequence, the task time can be increased by this sequence-independent part of subsequent setup times while the setup times need to be reduced accordingly by computing $t_i := t_i + \omega_i$, $\tau_{ij} := \tau_{ij} - \omega_i\ \forall j \in F_i^\tau$, and $\mu_{ij} := \mu_{ij} - \omega_i\ \forall j \in F_i^\mu$.

After this reduction step, there might remain sequence-independent parts α_i of preceding setup times for some tasks. For each such task i, the execution time can be increased further by setting $t_i := t_i + \alpha_i$, $\tau_{hi} := \tau_{hi} - \alpha_i\ \forall h \in P_i^\tau$, and $\mu_{hi} := \mu_{hi} - \alpha_i\ \forall h \in P_i^\mu$. Notice that these computations follow the same logic as the bound argument for the asymmetric traveling salesman problem proposed by Little et al. (1963).

In our example, we get $\omega = (1, 1, 1, 0, 1, 1, 1, 2)$ such that the execution times of the tasks 1, 2, 3, 5, 6, and 7 can be increased by 1 time unit while t_8 can be increased by 2. After reducing the setup time matrices, each column of either matrix contains at least one entry 0, so, all α-values are 0 and no further reductions possible.

The reduction step is performed before the lower bound value $\mathrm{LB1}^{\mathrm{fb}}_{\mathrm{AMP}}$ is calculated. This has two potential advantages: On the one hand, the AMP-bounds do not consider that each particular task has only one preceding and one succeeding setup as utilized in the reduction step. If only a subset of the tasks are responsible for the smallest setup times summed up by those bound arguments, the resulting lower bound values will be weak. On the other hand, increased task times represent better information for priority rules to select the right task for being assigned.

Note that, in many cases, the sums utilized in the bound arguments will be 0 after reduction. However, e.g., if only return times μ_{ii} are 0 and all other forward and backward setup times are positive, the reduction step will have no effect as $\alpha_i = \omega_i = 0\ \forall i$, whereas $\mathrm{LB1}^{\mathrm{fb}}_{\mathrm{AMP}}$ will take advantage of $\tau^q_{\mathrm{sum}} > 0$ (while $\mu^m_{\mathrm{sum}} = 0$).

7 Computational experiments

We describe a number of computational experiments and analyze their results to examine the performance of the procedures described so far and compare them to previous developments in the field.

7.1 Data sets

AMP-data set Andrés et al. (2008) defined a data set for SUALBSP (with unidirectional setup times only). This data set utilizes some precedence graphs contained in the well-known benchmark data set for SALBP-1 (cf. Scholl 1999, ch. 7.1. They systematically extracted eight (medium-sized) precedence graphs such that different order

strengths of the graphs and different numbers of tasks are included. Furthermore, they defined two settings with respect to the variability of setup times. In the setting called "low", setup times are randomly chosen from the interval $[0, 0.25 \cdot t_{\min}]$ with $t_{\min}$ denoting the minimal execution time of a task, while the times are randomly drawn from $[0, 0.75 \cdot t_{\min}]$ in the setting "high". For each precedence graph and each setting, ten instances are generated by randomly defining cycle times from the interval of cycle times occurring in the SALBP-data set. In order to transform these 160 instances to our version of SUALBSP, we have to set $\mu_{ij} := \tau_{ij}$ for all task pairs $(i, j) \in V \times V$.

MP-data set Martino and Pastor (2010) extended the AMP-data set by adding another eight precedence graphs from the SALBP-1 benchmark data set and defining two additional settings for a higher setup time level. In the setting called "low-med", setup times are randomly chosen from the interval $[0, 0.25 \cdot t_{\text{av}}]$ with t_{av} denoting the average execution time of a task. Similarly, a setting "high-med" draws randomly from $[0, 0.75 \cdot t_{\text{av}}]$. All together, 16 precedence graphs are combined with four settings. As again 10 instances are generated per combination, a total of 640 instances is contained in the MP-data set. Though it contains the AMP-data set completely, we show results for both data sets in order to be comparable with former experiments and to examine the effect of low and high setup times.

SBF-data sets In our opinion, the MP-data set has several limitations: (1) Instances were selected from the SALBP-1 benchmark data set rather arbitrarily. (2) No distinction between forward and backward setup times is made. (3) The triangle inequality is not observed in some cases which seems to be not realistic as insertion of a task in a sequence might reduce station time. (4) Martino and Pastor (2010) have fine-tuned their heuristics just for this data set. (5) Optimal solutions are known only for a part of the instances such that deviations from (rather weak) lower bounds need to be considered to measure solution quality. In order to remove these drawbacks, we define two settings SBF1 (removes drawbacks 1–4) and SBF2 (additionally removes drawback 5).

SBF1 consists of four data sets which are constructed as follows: As a basis, the entire SALBP-1 data set is taken such that each new data set contains 269 instances (having 7–297 tasks) with all data from SALBP-1 like cycle times, task times and precedence constraints. Different data sets are obtained by defining different maximal levels of the setup time. As in most cases there are rather small minimal task times and, furthermore, in practice setup times might be much larger than the time of such a minor task, we relate the setup times to the average task time t_{av} as done by Martino and Pastor (2010) in their extended set. We define $\alpha \cdot t_{\text{av}}$ as upper bound for the (forward) setup times to be generated. Four different data sets are obtained by setting α to the values 0.25, 0.50, 0.75, and 1.00. That is, average setup times are 12.5, 25, 37.5, and 50% of the average task time. This should cover almost any realistic case and allow for comparisons with the MP-data set.

Furthermore, setup times are constructed such that they hold the triangle inequality (1). In order to achieve realistic setup times, we subdivide each station into (fictitious) mounting positions $1, 2, \ldots, mp^{\max} = \lfloor \alpha \cdot t_{\text{av}} \rfloor$ in order to model setup times as walking times between real positions. Each task i gets assigned a mounting position mp_i randomly drawn from the interval $[0, \lfloor \alpha \cdot t_{\text{av}} \rfloor]$. The forward setup times are

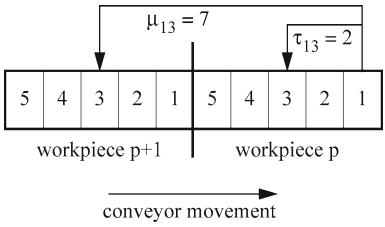


Fig. 4 Generation of setup times based on mounting positions

calculated as the absolute difference (distance) between mounting positions, i.e., $\tau_{ij} := |mp_i - mp_j|$. Backward setup times (return times) result from the fact that the worker has to walk to the next workpiece. If we assume that workpieces are arranged on the conveyor without space between them, return times have to be calculated by $\mu_{ij} := mp^{max} + mp_j - mp_i$. These calculations are visualized for $mp^{\max} = 5$ in Fig. 4. Our example of Fig. 2 has been generated in this manner by setting the position vector to $mp = (3, 4, 1, 2, 3, 4, 2, 1)$ with $mp^{\max} = 4$.

A second set SBF2 of four data sets is generated like SBF1, but it is taken care that at least one optimal SALBP-1 solution (with random precedence-feasible sequence within stations) remains feasible and, thus, constitutes an optimal solution to SUALBSP, also. This is achieved by setting all mounting positions of tasks assigned to the same station in this given solution to the same value (no forward setups required) and by restricting the backward setup times in each station to the idle time of the contained load. This is done such that the triangle inequality still holds.

All data sets are available for download at
http://www.assembly-line-balancing.de->Datasets.

7.2 Experimental settings

All experiments were run on a personal computer with quad-core AMD Phenom II X4 920 processor (four cores with 2.8 GHz clock speed each), and 3.0 GB of RAM. In order to speed-up computations, (up to) four parallel test runs were started at the four cores (without using parallel computation capabilities). As operating system, Windows XP Professional (32 bit) was used. The algorithms have been coded and compiled using Borland Delphi 7.0 as console applications.

We recoded the approaches of Andrés et al. (2008) and Martino and Pastor (2010) and implemented the new procedures described above. Additionally, the general-purpose ALBP-solver Avalanche (cf. Boysen and Fliedner 2008), which could be adapted easily to solve SUALBSP instances, was included in the tests. As a general benchmark, we finally implemented our model of Sect. 4 in Fico XPress-MP (Mosel 3.0, XPress optimizer 20.00.05). The XPress-model is initialized with an upper bound provided by applying a PRBP with priority rule MaxTMinSU. As it takes time to solve complex problems with off-the-shelf solvers like XPress-MP, we started it with a time limit of 100 s per instance.

Task-GRASP (called TG) of Andrés et al. (2008), described in Sect. 5.2, is restricted to 10 passes as suggested by the authors, whereas Avalanche (see Sect. 5.3) is restricted to 1,000 iterations with a population size of 25 sequence vectors (ants) as proposed

in Scholl et al. (2010). As the MP-rule of Martino and Pastor (2010) is a single-pass rule, it constructs a single solution (first step called MP) which is improved by local search after each assignment (both steps called MPLS) as described in Sect. 5.1.

In order to examine which of the components developed in this work and described in Sect. 6 are useful (and which not), we test a number of combinations each of which contains the Rule-GRASP solution framework (including the MP-rule deriving a complete solution) and the following further options:

F10	10 passes in forward direction
FB10	10 passes in forward and 10 passes in backward direction
FBC10	FB10 plus grouping graph approach of Sect. 6.4
FBR10	FB10 plus re-optimization as described in Sect. 6.2
FBCR10	FB10 plus grouping graph and re-optimization
FBCI10	FB10 plus grouping graph and fathoming as described in Sect. 6.3
FBRI10	FB10 plus re-optimization and fathoming
FBCRI10	FB10 plus grouping graph, re-optimization and fathoming
FBCRI100	FBCRI with 100 passes instead of 10
FBLS10	FB10 plus local search of Andrés et al. (2008) with first fit & extended move acceptance (Sect. 5.2)

In order to have identical and fair conditions and to enable procedures to examine potential optimality, the lower bounds and reduction process of Sect. 6.5 are applied before starting the heuristics in any case. The following measures are utilized to evaluate the algorithms:

rel.best	average relative deviation from the best upper bound found in the test (in %)
rel.LB	average relative deviation from best known lower bound (in %)
rel.opt	average relative deviation from the minimal number of stations m^* (only SBF2-data sets) (in %)
av.time	average computation time in seconds
#opt	the number of proven optima, i.e., procedure itself proves optimality due to UB = LB
#found	the number of optima found, i.e., UB $= m^*$ (only SBF2-data sets)
#best	the number of best solutions found (among the approaches within a test)

7.3 Experiments on the AMP- and MP-data sets

Table 2 summarizes the results obtained for the AMP-data set. There are considerable differences between the approaches concerning solution quality and/or computation time. The best relative deviations are obtained by FBCRI100 which combines all developed components. It finds the best solution for 157 of the 160 instances with moderate computation times of about 1.5 s per instance on average. If computation time is taken into account, too, the best compromise might be constituted by FBCR10 which is best in 151 out of 160 cases and takes only 0.24 s on average.

Table 2 Results of all procedures for AMP-data set (160 instances)

	MP	MPLS	TG	Aval.	Xpress	F10	FB10	FBC10	FBR10	FBCR10	FBCI10	FBRI10	FBCRI10	FBCRI100	FBLS10
rel.best	2.86	2.82	3.00	2.51	4.30	1.76	1.18	1.05	0.45	0.33	1.39	0.47	0.45	0.05	0.97
rel.LB	11.89	11.84	12.02	11.51	13.38	10.71	10.11	9.97	9.37	9.23	10.33	9.39	9.36	8.95	9.90
cpu	0.04	0.04	0.17	16.27	102.08	0.10	0.17	0.19	0.23	0.24	0.14	0.18	0.18	1.45	0.25
#opt	57	57	54	57	47	62	67	67	71	71	65	71	71	73	68
#best	115	115	106	113	94	127	134	139	145	151	132	144	145	157	135

Table 3 Results of all procedures for MP-data set (640 instances)

	MP	MPLS	TG	Aval.	Xpress	F10	FB10	FBC10	FBR10	FBCR10	FBCI10	FBRI10	FBCRI10	FBCRI100	FBLS10
rel.best	1.43	1.13	8.37	7.97	8.50	1.11	0.89	0.89	0.57	0.57	0.90	0.56	0.56	0.23	0.73
rel.LB	9.19	8.93	16.85	16.40	16.92	8.83	8.59	8.59	8.24	8.24	8.61	8.23	8.23	7.89	8.44
cpu	0.09	0.09	0.77	28.63	104.10	0.34	0.63	0.90	0.92	1.10	0.44	0.63	0.69	6.75	1.18
#opt	145	153	111	116	90	155	161	161	177	177	158	177	177	186	163
#best	398	412	262	261	222	438	479	489	569	583	475	570	571	627	485

In order to rank the new components concerning their contribution to good performance, it has to be stated that the Rule-GRASP mechanism, the additional backward planning and the re-optimization have most influence, while the grouping graph approach is successful only in a few cases but at very low cost in computation time. Fathoming significantly reduces computation time and allows for 100 (trial) passes (FBCRI100) in each direction without increasing computation time to 200 times the one of MP. However, in case of FBCI it is counterproductive as it reduces potential to apply the grouping graph approach. The enhanced local search of FBLS10 is not competitive to the other approaches.

As reported by Martino and Pastor (2010), MP performs very well considering the fact that only a single solution is computed. However, the AMP-data set is part of the data set MP is fine-tuned for. Here, the local search for re-optimization in MPLS is not very effective. The results of TG are worse than those of MP though ten solutions instead of one are computed.

Considering that Avalanche is a flexible multi-purpose procedure, its solution quality is acceptable but required time is large. Further, smaller tests show that computation times can be reduced without significant loss in solution quality, but we do without showing such results as the new procedures, nevertheless, outperform the other procedures. The performance of XPress-MP is very disappointing as its acceptable solution quality is mainly due to the initial heuristic solution computed. Even increasing computation times considerably has almost no effect. This shows the complexity of the problem and puts the good performance of the (other) heuristics in perspective.

When considering the MP-data set, we get similar results with even increased gap between well-performing procedures on the one hand and TG, Avalanche and XPress-MP on the other hand (Table 3). As MP has been fine-tuned to this very data set, its results are near-to-optimal though it requires negligible computation times. However, considerable improvements can be obtained by the same new procedures as before with FBRI10 providing a good compromise between solution quality and

Table 4 Results of all procedures for SBF1-data set (269 instances, $\alpha = 0.25$)

	MP	MPLS	TG	Aval.	Xpress	F10	FB10	FBC10	FBR10	FBCR10	FBCI10	FBRI10	FBCRI10	FBCRI100	FBLS10
rel.best	2.19	1.37	3.93	5.44	4.46	1.59	1.37	1.33	0.65	0.63	1.47	0.57	0.57	0.18	1.34
rel.LB	15.78	14.91	17.66	19.33	18.26	15.13	14.89	14.84	14.09	14.06	14.99	14.00	14.00	13.57	14.85
cpu	0.17	0.15	1.22	34.16	97.59	0.85	1.64	8.15	1.96	8.00	1.39	1.66	1.67	16.19	6.41
#opt	15	17	14	16	15	28	29	29	30	30	28	30	30	32	29
#best	153	191	112	102	102	172	182	186	233	235	179	234	234	260	184

Table 5 Results of all procedures for SBF1-data set (269 instances, $\alpha = 0.50$)

	MP	MPLS	TG	Aval.	Xpress	F10	FB10	FBC10	FBR10	FBCR10	FBCI10	FBRI10	FBCRI10	FBCRI100	FBLS10
rel.best	3.87	2.06	6.49	8.54	7.39	2.90	2.33	2.10	0.55	0.42	2.45	0.64	0.64	0.18	2.27
rel.LB	24.39	22.30	27.34	29.77	28.43	23.25	22.59	22.31	20.53	20.38	22.73	20.64	20.64	20.09	22.51
cpu	0.16	0.15	1.12	34.38	98.35	0.82	1.58	11.19	1.91	12.04	1.32	1.68	1.66	17.23	6.14
#opt	9	10	9	9	8	17	18	18	20	20	17	20	20	20	18
#best	110	177	82	67	67	126	145	155	230	239	141	224	224	258	148

Table 6 Results of all procedures for SBF1-data set (269 instances, $\alpha = 0.75$)

	MP	MPLS	TG	Aval.	Xpress	F10	FB10	FBC10	FBR10	FBCR10	FBCI10	FBRI10	FBCRI10	FBCRI100	FBLS10
rel.best	4.86	2.95	8.80	10.98	9.90	3.98	3.42	3.01	0.97	0.86	3.33	0.92	0.91	0.27	3.33
rel.LB	30.94	28.63	35.49	38.19	36.88	29.81	29.14	28.60	26.08	25.95	29.02	26.04	26.03	25.23	29.02
cpu	0.16	0.15	1.17	35.78	98.42	0.82	1.58	12.90	1.97	14.31	1.36	1.78	1.77	17.97	6.20
#opt	6	7	4	4	2	12	13	13	14	14	13	14	14	15	13
#best	89	136	64	50	53	102	114	128	213	223	113	212	213	257	116

computation time. The further quality improvement by FBCRI100 comes along with about tenfold computation time which, nevertheless, is moderate concerning the needs of practice.

7.4 Experiments on the SBF-data sets

In this section, we show and analyze the results obtained for the new SBF-data sets for different setup time levels and variability α. Tables 4, 5, 6 and 7 summarize the results obtained for the setting SBF1. The relative deviations from the lower bounds and the low numbers of proven optimal solutions show that these new data sets are much more challenging than the former ones. Though the rankings of the procedures are very similar to the aforementioned experiments, the relative differences in solution quality (rel.best) and computation times (cpu) become much larger. Again, FBRI10, and also FBCRI10, outperform the other procedures with small computation times and FBCRI100 provides further significant improvements which seem to justify increased computation times.

Considering the different values of α shows that increasing (average) setup times lead to considerably increased differences of the procedures' performance. In case of

Table 7 Results of all procedures for SBF1-data set (269 instances, $\alpha = 1.00$)

	MP	MPLS	TG	Aval.	Xpress	F10	FB10	FBC10	FBR10	FBCR10	FBCI10	FBRI10	FBCRI10	FBCRI100	FBLS10
rel.best	6.11	3.67	10.41	12.76	11.71	5.28	4.56	3.94	1.21	0.95	4.54	1.17	1.17	0.45	4.49
rel.LB	37.44	34.38	42.68	45.71	44.36	36.38	35.49	34.63	31.23	30.89	35.43	31.19	31.19	30.27	35.39
cpu	0.16	0.15	1.23	35.07	98.51	0.82	1.61	12.63	2.03	15.36	1.38	1.86	1.86	18.85	6.32
#opt	0	1	0	1	0	3	4	4	8	8	4	8	8	8	4
#best	71	112	61	44	45	79	89	104	195	215	88	198	198	244	91

Table 8 Results of all procedures for SBF2-data set (269 instances, $\alpha = 0.25$)

	MP	MPLS	TG	Aval.	Xpress	F10	FB10	FBC10	FBR10	FBCR10	FBCI10	FBRI10	FBCRI10	FBCRI100	FBLS10
rel.best	1.74	1.61	3.65	3.86	4.03	1.16	0.41	0.38	0.22	0.19	0.38	0.18	0.18	0.02	0.41
rel.opt	4.80	4.67	6.77	7.01	7.16	4.22	3.46	3.43	3.26	3.23	3.43	3.22	3.22	3.06	3.46
rel.LB	10.00	9.86	11.99	12.22	12.39	9.38	8.61	8.58	8.41	8.38	8.58	8.37	8.37	8.20	8.61
cpu	0.12	0.12	1.10	35.82	97.13	0.66	1.39	7.13	1.62	6.99	0.88	1.04	1.03	9.80	3.67
#opt	55	56	41	44	37	59	66	66	67	67	66	67	67	67	66
#found	108	109	84	91	79	123	134	136	137	139	135	139	139	144	134
#best	196	204	122	138	116	217	239	242	255	258	240	256	256	267	239

$\alpha = 0.25$ one might state that most of the heuristics can be useful solution approaches, whereas much larger differences are observed for larger values of α. Here, only the composite procedures which use the re-optimization component are competitive with FBCR10 being the best among all heuristics with ten passes. Comparing FBCR10 to FBR10 and FBC10 shows that it is the combination of re-optimization and grouping graph which provides this good solution quality. The versions with fathoming (I) achieve considerable reductions in computation times with a moderate deterioration in solution quality. So, it seems preferable to apply FBCRI100 instead of FBCR10 for larger α-values as it slightly improves solution quality at a modest increase in time.

The MP-rule still performs well considering the small computation times. This might result from those SALBP precedence graphs the former data sets and the SBF-data sets have in common. However, other cycle times and, more important, setup times have been chosen. The fine-tuning of the rule has obviously reached an astonishing robustness such that the rule can be recommended whenever a solution should be computed without any recognizable delay and setup times are relatively small.

The importance of re-optimizing (partial) sequences observed above is underlined by comparing the two versions of the MP-rule as the local search has a considerable potential of improvement which increases with α. Nevertheless, the MP-rule is clearly outperformed by the new composite heuristics. The other approaches (TG, Avalanche and XPress) are not competitive at all.

In order to examine the solution quality obtained by the procedures even more comprehensively, we repeat the experiment based on the setting SBF2. Due to modifying the data sets appropriately, the optimal solution values are known. This comes at the cost of small setup times between tasks assigned to the same station in the underlying SALBP-1 solution. As one might expect, the results summarized in Tables 8, 9, 10

Table 9 Results of all procedures for SBF2-data set (269 instances, $\alpha = 0.50$)

	MP	MPLS	TG	Aval.	Xpress	F10	FB10	FBC10	FBR10	FBCR10	FBCI10	FBRI10	FBCRI10	FBCRI100	FBLS10
rel.best	1.75	1.74	6.41	6.38	6.92	1.44	0.54	0.47	0.25	0.20	0.47	0.18	0.18	0.06	0.53
rel.opt	5.39	5.38	10.23	10.24	10.74	5.07	4.16	4.09	3.86	3.81	4.09	3.79	3.79	3.67	4.15
rel.LB	10.77	10.76	15.70	15.68	16.23	10.43	9.50	9.42	9.19	9.14	9.43	9.12	9.12	8.99	9.49
cpu	0.14	0.12	1.16	35.65	97.35	0.64	1.25	11.22	1.63	11.08	0.80	1.06	1.08	10.68	3.61
#opt	52	52	27	35	22	54	62	62	63	63	63	64	64	64	62
#found	104	104	60	75	50	112	124	124	126	126	124	127	127	130	124
#best	193	194	79	95	67	202	232	237	253	257	235	256	256	264	233

Table 10 Results of all procedures for SBF2-data set (269 instances, $\alpha = 0.75$)

	MP	MPLS	TG	Aval.	Xpress	F10	FB10	FBC10	FBR10	FBCR10	FBCI10	FBRI10	FBCRI10	FBCRI100	FBLS10
rel.best	2.46	2.19	9.56	8.63	9.94	2.09	0.71	0.64	0.25	0.19	0.62	0.28	0.28	0.10	0.68
rel.opt	6.61	6.32	14.03	13.11	14.40	6.23	4.82	4.75	4.33	4.27	4.73	4.37	4.37	4.18	4.78
rel.LB	12.01	11.71	19.58	18.62	20.01	11.61	10.18	10.10	9.68	9.61	10.08	9.71	9.71	9.52	10.14
cpu	0.13	0.12	1.24	35.92	97.87	0.69	1.39	15.48	1.83	16.83	0.86	1.08	1.09	10.99	3.67
#opt	45	45	22	32	19	48	60	60	60	60	60	60	60	61	60
#found	91	92	46	67	37	99	116	116	121	121	116	122	121	124	116
#best	171	184	56	83	44	182	221	224	256	259	224	253	254	261	222

Table 11 Results of all procedures for SBF2-data set (269 instances, $\alpha = 1.00$)

	MP	MPLS	TG	Aval.	Xpress	F10	FB10	FBC10	FBR10	FBCR10	FBCI10	FBRI10	FBCRI10	FBCRI100	FBLS10
rel.best	2.83	2.72	12.24	8.41	11.61	2.28	1.09	1.02	0.57	0.50	1.01	0.53	0.50	0.14	1.01
rel.opt	7.19	7.07	17.10	13.11	16.39	6.64	5.40	5.33	4.86	4.78	5.32	4.81	4.79	4.42	5.32
rel.LB	12.64	12.51	22.72	18.62	22.04	12.05	10.78	10.70	10.22	10.13	10.69	10.17	10.14	9.76	10.70
cpu	0.12	0.12	1.29	35.92	97.54	0.63	1.24	17.98	1.79	17.16	0.83	1.18	1.20	11.65	3.52
#opt	45	44	22	32	21	49	55	55	58	58	55	58	58	61	56
#found	84	85	43	67	36	95	111	111	115	116	113	115	116	122	112
#best	150	157	48	82	43	162	204	205	241	246	207	244	245	260	209

and 11 indeed show, that this way of generating data sets simplifies finding a good or even optimal solution. This is particularly true for construction heuristics calculating priority values based on (minimal) setup times. Nevertheless, the findings are very similar to those described for SBF1 but at a lower level of differences.

To examine the larger deviations in case of the SBF1-data sets, we compare the average relative deviations from the lower bounds (rel.LB) to those from optimality (rel.opt) in case of the SBF2-data sets. This comparison shows that more than half of this deviation is due to rather weak lower bounds. So, one can assume seriously that the relative deviations from optimality in case of the SBF1-data sets will also be much smaller than indicated by rel.LB.

The fathoming component of the composite heuristics again appears to be important as it avoids unnecessary computations. This becomes obvious by comparing FBCR10 and FBCRI10, where almost identical solution quality is obtained with the

latter procedure taking roughly 10% of the time. Now, FBCRI10 seems to be the best compromise considering quality and time as it requires about 1 s on average for small optimality gaps of about 3–5% even in case of large and considerably varying setup times, where some of the procedures have deviations of 15% and more. When about 10 s are invested per instance, FBCRI100 almost in every case finds the best solution.

To summarize the findings of our performance experiments, it can be stated that the new composite heuristic seems to be a well-suited compromise for finding good solutions for SUALBSP in short time as required in real-world planning environments (cf. Sect. 1). All components of the new procedure (Rule-GRASP, different planning directions, fathoming, grouping graph and re-optimization) seem to be useful as they contribute to improving solutions provided that they are applied in a time-saving and problem adequate manner. This tradeoff between effectiveness and computation time can be demonstrated for the grouping graph approach which is successful in improving solutions very seldom but does require negligible time.

7.5 On the importance of SUALBSP

We conduct a final experiment to find out whether the additional effort of defining and solving SUALBSP instances might indeed be useful in practice of assembly line balancing. As an alternative, the problem could be modeled and solved by considering setup times in a simpler way as is usually done in practice. Some major car producers, for instance, simply approximate sequence-dependent setup times and incorporate the results into the task times.

In doing so, the planning team needs to strike a fragile balance between an underestimation of setups, which leads to infeasible line balances, and an overestimation of setups, which leads to an allocation of excessive resources. In practice, this trade-off is usually considered by an iterative planning process which successively re-evaluates line balances with regard to feasibility and resource utilization and re-optimizes subsequences of tasks where setups have been estimated incorrectly.

This planning procedure is very time consuming and prone to be caught in local suboptima. We simulate this approach by defining several ways of transforming a SUALBSP instance into a SALBP-1 instance which can be solved by the well-known exact solution procedure SALOME (Scholl and Klein 1997, 1999). The results are examined with respect to feasibility and capacity requirement of the production system, i.e., number of stations, which constitute surrogate values for the required planning effort to improve such a solution. We consider the following approaches to cope with setup times without using SUALBSP (decimal values of task time t'_i for $i \in V$ are rounded to the next integer):

1. **NoS:** The setup times can be *ignored* completely defining a SALBP-1 instance with the original task times, i.e., $t'_i := t_i$. This procedure assumes that setup times are negligible or sequence-independent (and so can be integrated in task times) or reflects that the planner ignores the phenomenon. While this is not so much of a practicable approximation, we include this simple rule in order to better estimate the effects of setups on solution feasibility.

2. **MiS:** Task times are increased by minimal setup time defined as $\tau_i^{\min} := \min\{\min\{\tau_{ij} \mid j \in F_i^{\tau}\}, \min\{\mu_{ij} \mid j \in F_i^{\mu}\}$, i.e., $t_i' := t_i + \tau_i^{\min}$. This rule can be seen as a form of optimistic planning, in the sense that it systematically underestimates setup times (by setting them to their lower bounds) and, thus, bears a considerable risk of generating infeasible line balances. Nonetheless, it keeps resource utilization at a maximum. It is not necessary to determine the entire matrices of forward and backward setup times but to estimate the required minimal ones only.
3. **MeIS:** The task time is increased by its average (mean) setup time defined as $\overline{\tau}_i := \frac{\sum_{j \in F_i^{\tau}} \tau_{ij} + \sum_{j \in F_i^{\mu}} \mu_{ij}}{|F_i^{\tau}| + |F_i^{\mu}|}$, i.e., $t_i' := t_i + \overline{\tau}_i$. This common planning approach is neutral, in that it seeks a balance between capacity overloads and capacity utilization. It is typical in cases where uncertainty of a certain aspect has to be handled and measuring exact values is thought to be too expensive and difficult. Furthermore, it reflects that small and large setup times have to be mixed in a (good) feasible solution.
4. **MeS:** Each task time is increased by the mean of all setup times defined as $\overline{\tau} := \frac{\sum_{i \in V}(\sum_{j \in F_i^{\tau}} \tau_{ij} + \sum_{j \in F_i^{\mu}} \mu_{ij})}{\sum_{i \in V}(|F_i^{\tau}| + |F_i^{\mu}|)}$, i.e., $t_i' := t_i + \overline{\tau}$. Unlike the previous rule, average setup times are distributed over all tasks, in order to prevent that individual tasks receive unduly high task times. It is based on the assumption that any given combination of tasks at a station is expected to have the same setup intensity as total population of tasks. This approach simplifies data gathering as it is sufficient to estimate a single mean setup time value.
5. **MeP:** The task time is increased by the (time) distance to and from the mean mounting position defined as $\overline{\tau'}_i := 2 \cdot |mp_i - \frac{1+mp^{\max}}{2}|$ (cf. Sect. 7.1), i.e., $t_i' := t_i + \overline{\tau'}_i$. This rule constitutes a more direct approach to estimate setup times based on walking distances. The mean mounting position can be interpreted as the starting point of all operations to which a worker needs to return on average after the execution of a task. So, the sequence-dependent setup times are made sequence-independent. This approach, however, ignores walking distances to the next workpiece so setup times are slightly underestimated.
6. **MxS:** The task time is increased by the maximal setup time defined as $\tau_i^{\max} := \max\{\max\{\tau_{ij} \mid j \in F_i^{\tau}\}, \max\{\mu_{ij} \mid j \in F_i^{\mu}\}$, i.e., $t_i' := t_i + \tau_i^{\max}$. This is a very risk aversive approach that guarantees feasible solutions but might overestimate real setup times and, thus, capacity requirements considerably.

We apply these approaches to the SBF1-data sets, i.e., six SALBP-1 instances have to be defined and solved by SALOME for each of the $4 \cdot 269 = 1{,}076$ instances of the four data sets. The resulting line balances are evaluated with respect to the real task times and setup times of the original SUALBSP instances. Besides the number of stations required, it is to be examined whether the station loads hold the cycle time restrictions. The following measures are used to compare the different approaches including deviations of the SALBP-based approaches from the best solution found by any heuristic as described in Sect. 7.4 on the one hand and the lower

Table 12 Results of the task time adaptation methods for SBF1-data sets (average on all 1,076 instances)

Total	NoS	MiS	MeIS	MeS	MeP	MxS
Infeas	80.85	77.91	6.03	26.30	31.59	0.00
Overtime	18.35	16.48	0.37	1.78	2.08	0.00
Increase	37.02	35.99	5.25	11.23	9.21	0.00
LBdev	3.09	5.48	41.08	27.61	26.69	77.90
Bestdev	−14.46	−12.49	15.94	4.91	4.38	45.87

bound on the other hand, because optimal solutions are not known for the SBF1-data sets:

Infeas	average portion of infeasible station loads in %
Overtime	average extent of violating the cycle time relative to total capacity $m \cdot c$ in %
Increase	average relative increase of the cycle time to achieve feasible line balances in %
LBdev	average relative deviation from lower bound on number of stations in %
Bestdev	average relative deviation from best known number of stations in %

A summary of the overall results is given in Table 12. Obviously, only MxS is able to guarantee feasible solutions. However, this comes at the cost of an average overcapacity of at least 46%, i.e., the number of stations required to achieve feasible station loads for sure is 46% larger than the best known number of stations computed by our SUALBSP heuristics (up to 78% compared to the lower bounds). This is absolutely unacceptable. All other approaches are far from generating feasible solutions as between 6 and 81% of the station loads exceed the cycle time. The least portion of unfeasible loads and least overtime percentage are caused by MeIS. However, this requires an excess capacity of 15–41%, i.e., the optimal number of stations must be exceeded by at least 15% to achieve almost feasible solutions. The other methods are even worse with ignoring the setup times (NoS) being the worst choice as the solutions completely differ from what is needed. Though MeS and MeP even require considerably more stations than needed, they are also far from finding feasible solutions.

A more detailed analysis shows that even for small setup times ($\alpha = 0.25$) none of the methods is competitive. These bad results are due to having relaxed the most important aspect of the problem. Thus, there is no useful alternative to using SUALBSP to model assembly lines where sequence-dependent setup times are relevant.

8 Conclusions

In this paper, we have studied and modeled the assembly line balancing and scheduling problem with setup times. We provide an extended yet more efficient mathematical model in comparison with prior research. We further propose a toolbox comprised of

several heuristic components, which can be combined as required to yield fast and/or high-quality solutions for instances of real-world size. The results of an extensive computational study demonstrate that the new heuristics dominate former approaches with regard to solution quality and computation times. Nevertheless, the computational experiments show that there is a need for improved solution procedures, in particular, in case of large setup times. A promising direction for future research could be the adaptation of sophisticated meta heuristics, such as tabu search or genetic algorithms, in order to seek a better trade-off between solution quality and computational time. Furthermore, research effort is required to analyze the mathematical structure of the problem more deeply as it contains an NP-hard subproblem in every station, namely, a sequencing problem of the traveling salesman type. There is a need for improved lower bounds and exact solution procedures, which are at least useful for improved testing of solution approaches.

Furthermore, it is required to consider additional aspects like the positioning of material boxes. This problem is interrelated with the problem considered here as the positions of boxes determine the distances to walk when fetching parts. The other way round, the sequence influences the optimal positions of boxes.

Another interrelated aspect is ergonomics. On the one hand, ergonomic restrictions introduce (additional) setups as tools like lifting or fixing devices have to be mounted. On the other hand, they might reduce task times. Furthermore, there is a potential advantage of setup operations (in particular, walking) as they enable workers to relax between two demanding tasks. So, setups seem to be useful for modeling dynamic aspects of ergonomics which receives increasing attention in (automotive) industry.

Acknowledgments The authors are indebted to two anonymous referees and the editor for their remarks on former versions of the paper which helped to improve readability and to extend the computational experiments in a useful way.

References

Allahverdi A, Ng C, Cheng T, Kovalyov M (2008) A survey of scheduling problems with setup times or costs. Eur J Oper Res 187(3):985–1032

Andrés C, Miralles C, Pastor R (2008) Balancing and scheduling tasks in assembly lines with sequence-dependent setup times. Eur J Oper Res 187(3):1212–1223

Arcus A (1966) A computer method of sequencing operations for assembly lines. Int J Prod Res 4:259–277

Barnes R (1959) Motion and time study. Wiley, New York

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

Baybars I (1986) A survey of exact algorithms for the simple assembly line balancing problem. Manag 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

Becker C, Scholl A (2009) Balancing assembly lines with variable parallel workplaces: problem definition and effective solution procedure. Eur J Oper Res 199:359–374

Boctor F (1995) A multiple-rule heuristic for assembly line balancing. Eur J Oper Res Soc 46:62–69

Boysen N, Fliedner M (2008) A versatile algorithm for assembly line balancing. Eur J Oper Res 184:39–55

Boysen N, Fliedner M, Scholl A (2007) A classification of assembly line balancing problems. Eur J Oper Res 183:674–693

Boysen N, Scholl A (2009) A general solution framework for component commonality problems. BuR Business Res 2(1):86–106

Dorigo M, Blum C (2005) Ant colony optimization theory: A survey. Theoret Comput Sci 344:243–278
Dorigo M, Stützle T (2003) The ant colony optimization metaheuristic: algorithms, applications, and advances, vol 57. Springer, New York, pp 250–285
Easton F, Faaland B, Klastorin T, Schmitt T (1989) Improved network based algorithms for the assembly line balancing problem. Int J Prod Res 27:1901–1915
Erel E, Sarin S (1998) A survey of the assembly line balancing procedures. Prod Plann Control 9:414–434
Helgeson W, Birnie D (1961) Assembly line balancing using the ranked positional weight technique. J Ind Eng 12(6):394–398
Jackson J (1956) A computing procedure for a line balancing problem. Manag Sci 2:261–271
Kanawaty G (1992) Introduction to work study. 4th Int Labour Office, Geneva
Klein M (1963) On assembly line balancing. Oper Res 11:274–281
Klein R, Scholl A (1999) Computing lower bounds by destructive improvement—an application to resource-constrained project scheduling. Eur J Oper Res 112:322–346
Little J, Murty K, Sweeney D, Karel C (1963) An algorithm for the traveling salesman problem. Oper Res 11:972–989
Martino L, Pastor R (2010) Heuristic procedures for solving the general assembly line balancing problem with setups. Int J Prod Res 48(6), 1787–1804
Miller C, Tucker A, Zemlin R (1960) Integer programming formulation of the traveling salesman problem. J Assoc Comput Mach 7:326–329
Nelder J, Mead R (1965) A simplex method for function minimization. Comput J 7:308–313
Pastor R, Andrés C, Miralles C (2010) Corrigendum to Balancing and scheduling tasks in assembly lines with sequence-dependent setup. Eur J Oper Res 201:336–336 (European J Operational Research 187 (2008) 1212–1223)
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
Resende M, Ribeiro C (2003) Greedy randomized adaptive search procedures. In: Glover F, Kochenberger G (eds) Handbook of Metaheuristics. Kluwer Academic Publishers, Dordrecht, pp 219–249
Scholl A (1999) Balancing and sequencing of 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, Boysen N, Fliedner M (2008) The sequence-dependent assembly line balancing problem. Oper Res Spectrum 30:579–609
Scholl A, Fliedner M, Boysen N (2010) Absalom: Balancing assembly lines with assignment restrictions. Eur J Oper Res 200(3):688–701
Scholl A, Klein R (1997) Salome: a bidirectional branch and bound procedure for assembly line balancing. INFORMS J on 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
Talbot F, Patterson J, Gehrlein W (1986) A comparative evaluation of heuristic line balancing techniques. Manag Sci 32:430–454
Wilhelm W (1999) A column-generation approach for the assembly system design problem with tool changes. Int J Flexible Manuf Syst 11(2):177–205